MANIPULATING STATE SPACE DISTRIBUTIONS FOR SAMPLE-EFFICIENT IMITATION-LEARNING

A Dissertation
Presented to
The Academic Faculty

By

Yannick Schroecker

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

May 2020

Copyright © Yannick Schroecker 2020

MANIPULATING STATE SPACE DISTRIBUTIONS FOR SAMPLE-EFFICIENT IMITATION-LEARNING

Approved by:

Dr. Charles Isbell, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Irfan Essa
School of Interactive Computing
Georgia Institute of Technology

Dr. Sonia Chernova
School of Interactive Computing
Georgia Institute of Technology

Dr. Byron Boots
School of Interactive Computing
Georgia Institute of Technology

Dr. Nando de Freitas
Deepmind Technologies

Date Approved: February 7, 2020

ACKNOWLEDGMENTS

This work would not have been possible without the involvement and support of many people who helped me along the way.

I would like to thank my thesis advisor, Charles Isbell, for his incredible support that allowed me to pursue the research directions which I found most interesting. Besides his advice, his encouragement and optimism helped me greatly when I was struggling with research challenges.

I would like to thank my committee members, Byron Boots, Sonia Chernova, Irfan Essa and Nando de Freitas for their time and their useful feedback that has helped me create this document. During my first 1.5 years, I have had the chance to work with Andrea Thomaz whose guidance and advice helped me when I was just starting out. I was fortunate to be able to collaborate with Heni Ben Amor on my first project during the first year of my PhD. Heni had joined Georgia Tech as a post-doc and was a fantastic mentor to me during that time.

After my first three semesters, I fully joined the pfunk lab where I found some of my closest friends during my PhD and in general. Himanshu Sahni was the first friend I made in this group and is responsible for many of the great experiences I had while being a grad student. Ashley Edwards was bringing the group closer together. Most recently, but perhaps inevitably, Shray Bansal switched to our lab. Long before this happened, Shray has been a very close friend to me and our many discussions late into the night made me think deeply about my objectives in life and about the meaning of the word ”risotto”. I am also grateful to have been a part of this lab while Pushkar Kohle was still here. Most recently, Vitaly Marin joined the lab as a masters student and deserves an acknowledgment. I am also glad to have still overlapped with Kaushik Subramanian. Before joining pfunk, I felt welcome as a member of the SIM lab and for that, I would like to thank Vivian Chu, Tesca Fitzgerald, Baris Akgun, Crystal Chao and Eric Huang.

My time at Georgia Tech has been a joy due to the friends I met here, including Amrita Gupta, Ashwath Kumar, Qi Zhang, Safoora Yousefi, Amirreza Shaban, Sasha Lambert, Jon Balloch, Brian Goldfain, and Steven Hickson.

I would like to thank Kalesha Bullard for all the great times we have had together, for pushing me to new experiences, for always being supportive, for making my time at work more entertaining, and for making sure I maintain an appropriate sense of cynicism towards the PhD.

Finally, I would like to thank my parents for encouraging me to explore this path and supporting me all the way.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Imitation learning has emerged as one of the most effective approaches to train agents to act intelligently in unstructured and unknown domains. On its own or in combination with reinforcement learning, it enables agents to copy the expert's behavior and to solve complex, long-term decision making problems. However, to utilize demonstrations effectively and learn from a finite amount of data, the agent needs to develop an understanding of the environment. This thesis investigates estimators of the state-distribution gradient as a means to influence which states the agent will see and thereby guide it to imitate the expert's behavior. Furthermore, this thesis will show that approaches which reason over future states in this way are able to learn from sparse signals and thus provide a way to effectively program agents. Specifically, this dissertation aims to validate the following thesis statement: **Exploiting inherent structure in Markov chain stationary distributions allows learning agents to reason about likely future observations, and enables robust and efficient imitation learning, providing an effective and interactive way to teach agents from minimal demonstrations.**

CHAPTER 1
INTRODUCTION

1.1 Motivation

The development of agents that are able to act intelligently in unknown and unstructured environments using as little prior knowledge as is possible is a central promise of the field of artificial intelligence. To fulfill it, agents need to be able to learn to act in environments that are sequential and unknown. Over the course of the last decades, two central paradigms have emerged that aim to address the sequential decision making problem in the general case. Reinforcement learning promises to enable agents to learn optimal behavior from their own experience, where the only requirement is a numerical reward signal that encodes a notion of optimality, measuring desirability of every event in the agent's life span. While this approach is highly general and has shown many great successes over the years on a variety of domains, such as video games (e.g. Mnih *et al.* [68] and Espeholt *et al.* [24]), robotics (e.g. Kober *et al.* [49], Levine *et al.* [59], and Andrychowicz *et al.* [4]) or solving the long-standing problem of the game of Go [102], learning from a finite amount of interaction with the environment remains a difficult challenge as the provided reward often only provides a sparse signal to the agent. Even when existing learning algorithms are sufficiently efficient, specifying a suitable reward function that enables efficient learning by hand is often difficult. Imitation learning provides an avenue to tackle this challenge by allowing agents to learn from human teachers, which constitutes a natural way for experts to describe the desired behavior and provides an efficient learning signal for the agent. It is thus no surprise that imitation learning has enabled great successes on robotic [17] as well as software domains (e.g. Aytar *et al.* [8]). Besides providing a highly effective paradigm for training agents by itself, it is often also possible to be used to help a

reinforcement learning agent learn near-optimal policies where learning from reward alone requires a prohibitively large number of environment interactions and computational effort (e.g. Peters and Schaal [83], Večerík *et al.* [121], Sun *et al.* [106], Zhu *et al.* [127], and Chernova and Thomaz [17]).

In imitation learning, each demonstration provides additional information to the agent. A significant amount of information can be encoded in each demonstration sample. For example, if the agent's goal is to simply reach a particular point in its observation space, a single demonstrated pose at the goal-location can be sufficient for the agent to understand its goal. Despite the amount of data encoded in each demonstrated pose, a single sample in itself will not be sufficient for the agent to learn how to achieve this pose, even if it is knows that it should reproduce it. Without a better understanding of the environment, the agent has no way of knowing which obstacles may be in its way or even what the effect of its own actuations are on its movement toward or away from the desired pose. It is for this reason that naive approaches to imitation learning often require a large amount of demonstration data, which can be difficult to collect and expensive for an expert teacher to provide.

As a central component of this dissertation, we introduce and explore novel approaches to learn from minimal amounts of exploration data. We will show that this enables the agent to solve complex problems, even when expert data is sparse, and at times enables the expert to teach the agent using incomplete data when recording good, complete demonstrations is prohibitively difficult. In principle, the imitation learning problem could be seen as a supervised learning problem, where the demonstrations are used to learn a mapping from observed states to actions. This solution approach is known as behavioral cloning. However, it has long been known that the sequential structure of the task admits more effective solutions [91, 73]. The amount of demonstration data behavioral cloning usually requires is considerable.

To achieve more sample-efficient and robust imitation, the agent needs to be able to reason about future states it will observe and be able to reason about how to change said

distribution to reproduce observations like those it has seen in the given demonstrations. This requires the agent to recover when exact imitation is impossible and it is deviating from the desired path. In prior work, this reasoning is often present but highly implicit, e.g. through learning a fixed [73, 129] or changing [39] reward function that acts as a surrogate loss, or makes additional assumptions about the problem domain, e.g. Pathak *et al.* [78] and Peng *et al.* [80]. This dissertation aims to, first, highlight that reasoning over future states explicitly enables sample-efficient and effective imitation learning and, second, introduces a methodology that enables agents to perform this reasoning explicitly and in the general case with minimal assumptions and knowledge about the domain.

1.2 Thesis Statement

Specifically, this dissertation aims to make and validate two central and highly connected claims:

1. Reasoning over the agent's future observations enables effective imitation learning from small amounts of demonstration data and provides a natural and effective way for an expert demonstrator to encode the desired behavior and teach the agent.

2. Exploiting intrinsic structure in the distribution of future observation enables us to reason over changes to this distribution and in turn enables the development of efficient imitation learning algorithms. This structure follows based on basic assumptions that are commonly made in imitation learning and are easy to fulfill.

The combination of those claims, both of which will be addressed and validated in this document, leads us to the central statement of this thesis:

Exploiting inherent structure in Markov chain stationary distributions allows learning agents to reason about likely future observations, and enables robust and efficient imitation learning, providing an effective and interactive way to teach agents from minimal demonstrations.

1.3 Contributions

To validate the central claim in its two parts, this dissertation makes the following contributions:

1. This dissertation introduces a novel approach to utilize minimal and sparse demonstrations in the form of via-points to guide a trajectory-based reinforcement learning system and to find a near-optimal solution to robotic manipulation task. This contribution chiefly tackles the first claim of the thesis statement by considering a special case scenario in which approximate reasoning over future observations is straightforward and shows how such reasoning can enable efficient imitation-learning in a composite imitation and reinforcement learning system.

2. This dissertation introduces an approach to utilize the recursive structure of the stationary distribution in Markov chains to describe how changes to the policy affect future observations and to derive a novel view on the imitation learning problem. This contribution lays the foundation for reasoning over future states in the general case and lays the foundation to validate the second claim.

3. This dissertation introduces a novel imitation learning algorithm that utilizes said structure, as well as temporal difference learning techniques found in reinforcement learning, in order to learn from sparse demonstrations in the general case. This contribution thus utilizes insights derived from the previous contribution to validate the second claim of the thesis statement.

4. This dissertation furthermore introduces an approach that utilizes said structure, as well as recent techniques developed in generative modeling, to learn from sparse demonstrations, as well as sparse environment interactions. This contribution further demonstrates the principle for achieving effective and efficient imitation learning and shows applicability to real world task domains.

5. This dissertation introduces an approach to unbiased and efficient goal-conditioned reinforcement learning, teaching the agent to reach arbitrary states by using aforementioned generative modeling techniques. In the context of the thesis statement, this contribution chiefly lays the technical foundation to integrate variance-reduction methods from reinforcement learning into a framework based on long-term predictive generative models.

6. This dissertation introduces an approach to combine said generative modeling techniques with variance-reduction methods found in reinforcement learning to enable effective imitation learning in highly complex domains. This contribution broadens the range of tasks on which the findings can be applied.

1.4 Outline

Before delving into the specific contributions, we will first formalize the reinforcement- and imitation learning problems, and review the relevant background literature to further situate this work in Chapters 2 and 3. Chapter 4 introduces a complete reinforcement- and imitation learning system in the special case of trajectory learning where assumptions about future observations can be made. Moving toward the more general case, we then introduce a novel imitation learning algorithm that is able to reason over future states in the more general case in Chapter 5. Chapter 6 reviews recent advances in generative models and introduces the concept of long-term generative models. This concept is used throughout the remaining chapters. Building on it and the findings in Chapter 5, Chapter 7 introduces an improved imitation learning algorithm that is more readily applicable to real world domains. In Chapter 8, we draw a connection to goal-conditioned reinforcement learning and introduce a new, efficient algorithm to this field. Combining these findings with the results from Chapter 7, we will then introduce a final, state-of-the-art imitation learning method in Chapter 9.

CHAPTER 2
MARKOV DECISION PROCESSES

Imitation learning, the primary concern of this thesis, belongs to the larger field of sequential decision making. Formally, sequential decision making problems are commonly described as Markov Decision Processes (MDPs) [86] and understanding the formalism is helpful to reason about the imitation learning problem. Here, we will define the notation and briefly review solution approaches based on reinforcement-learning. Both will be used in the coming chapters. The primary goal of this section is to lay the foundation for the algorithms derived in this thesis. We will leave a more complete review of the imitation learning landscape, situating the work described in this thesis, for the next chapter.

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) [86] provide us with a way to describe sequential decision making problems, making one central assumption to simplify the problem. In an MDP, the dynamics are assumed to be Markovian, i.e. the state of the system depends solely on the state and actions taken in the previous time-step. The result is that the problem can be described in terms of the state and action spaces, the transition dynamics and an initial state. Usually, the definition includes the reward function; however, as the reward is not present in an imitation learning context, we leave that for the next section where we discuss the reinforcement learning formalism[1].

Given state and action sets $\mathcal{S}, \mathcal{A}$, the agent is observing states $s \in \mathcal{S}$ and taking actions $a \in \mathcal{A}$. To clarify notation, here, we use s and a to refer to observed states and actions in general, s, a, s' to refer to a transition sequence of state, action and next state, $s^{(i)}, a^{(i)}$, i.e. superscripts, to refer to specific instances in a batch or demonstration, s_t, a_t, i.e. subscripts,

[1]Markov Decision Processes without a pre-defined reward function are sometimes referred to as MDP\R

to indicate temporal sequences, and $\overline{s}, \overline{a}$ to indicate desired or demonstrated states and actions.

Starting in a state sampled from a distribution of initial states, $s_0 \sim p_0$, the agent repeatedly takes actions a and observes the transition from state s to another state s'. In an MDP, the observed transitions are guided by environment dynamics that are assumed to be Markovian. The probability of transitioning from state s to state s' by taking action a is denoted as $p(s_{t+1} = s' | s_t = s, a_t = a)$.

The agent's behavior is defined by a policy, here parameterized by θ. Such a policy can be a probability distribution $\pi(a|s)$ or a deterministic function $\mu(s)$ (following the notation in Silver *et al.* [103], we use μ to refer to deterministic policies). Furthermore, it can be stationary ($\pi_\theta(a|s)$) or time-dependent ($\pi_\theta(a|s,t)$). In this thesis, policy representations will be either

1. Stationary, tabular policies which map finite sets of states to probabilities for a finite set of actions.

2. Neural networks [11], which, being universal function approximators, are capable of representing arbitrary behavior on arbitrary state-action spaces and are used to represent stationary policies as well.

3. Trajectory representations and, in particular, Dynamic Movement Primitives [42] which are time-dependent and will be introduced in more detail in Section 4.2.3.

Following a policy π induces a Markov state-action process with the probability of seeing a state s at time-step t being denoted as $d_t^\pi(s)$. If the Markov process is ergodic, which is usually assumed and easily ensured, each policy induces a unique stationary distribution of observed states

$$d^\pi(s) = \lim_{t \to \infty} d_t^\pi(s)$$

as well as a stationary joint state-action distribution

$$\rho^{\pi}(s, a) := d^{\pi}(s)\pi(a|s).$$

Long-term transition dynamics $p_{\pi}(s_{t+j}|s_t, a_t)$ are also dependent on the policy (as we only condition on the first action in the sequence leading from s_t to s_{t+j}). Of particular interest to this thesis will also be the time-reversed state-action process. The time-reversed transition probabilities are not, in general, stationary and we define:

$$q_t^{\pi}(s_t = s, a_t = a|s_{t+1} = s') = d_{t+1}(s')^{-1}d_t(s)\pi(a|s)p(s_{t+1} = s'|s_t = s, a_t = a) \quad (2.1)$$

We furthermore define $q^{\pi}(s_t = s, a_t = a|s_{t+1} = s') := \lim_{t\to\infty} q_t^{\pi}(s_t = s, a_t = a|s_{t+1} = s')$ as the transition probabilities for the time-reverse state-action process in the limit. This refers to the stationary transition probabilities assuming an infinite amount of prior steps and will be useful to reason about d^{π}. Often, a MDP will also have a set of terminal states, i.e. states after which the behavior of the agent is irrelevant and the agent may be reset. When reasoning about stationary distributions, such states are best modeled to transition back to initial states. This makes it such that the stationary distribution will be equal to the empirical distribution of states one is expected to observe when following a policy until a terminal state is reached. In that case q_t and q can assumed to be equal. If there are no terminal states, t tends to be large for observed states and we may assume $q(s_t = s, a_t = a|s_{t+1} = s') \approx q_t(s_t = s, a_t = a|s_{t+1} = s')$. Finally, we extend the notation for time-reversed transitions to multi-step transitions by writing $q^{\pi}(s_t = s, a_t = a|s_{t+j} = s')$.

2.2 Reinforcement Learning

2.2.1 Problem set-up

In reinforcement learning, the goal is to learn a policy which behaves optimally by maximizing the expected, long-term reward under a given reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The most commonly used objective is to maximize the expected discounted cumulative reward using a given discount factor γ, i.e. the optimal policy is defined as

$$\begin{aligned} \pi^* &:= \text{argmax}_\pi J(\pi); \\ J(\pi) &:= E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right], \end{aligned} \tag{2.2}$$

where we write E_π to denote the expectation over state-action sequences found when following policy π. As γ approaches 1, the optimal policy under this objective furthermore also maximizes the average reward objective, i.e. it maximizes the expected reward for all time-steps [86].

In this work, we will be concerned with model-free reinforcement learning, i.e. algorithms which maximize the reinforcement learning objective without explicitly learning a transition model $p(s_{t+1}|s_t, a_t)$. In particular, one can divide most model-free reinforcement learning algorithms into three classes: value-based methods, policy gradient methods and direct policy search methods. We will focus here on introducing these approaches in the context in which they relate to the work introduced in this thesis.

2.2.2 Temporal Difference Learning

A central concept to both, value-based as well as policy-gradient methods is the concept of a value function. The value function is defined as

$$V^\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s \right]. \tag{2.3}$$

Often it is useful to consider the Q-value function instead which evaluates the value of a state-action pair:

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right]. \tag{2.4}$$

Temporal difference learning [107] enables us to estimate either value function from s_t, a_t, s_{t+1} triples which have been obtained by following the policy π. In particular, when representing the value function as a parametric function V^{π}_{ω}, the basic temporal difference update rule is given as

$$V^{\pi}_{\omega^{(k+1)}}(s_t) = V^{\pi}_{\omega^{(k)}}(s_t) + \alpha \left(R(s_t, a_t) + \gamma V^{\pi}_{\omega^{(k)}}(s_{t+1}) - V^{\pi}_{\omega^{(k)}}(s_t)\right) \nabla_{\omega} V^{\pi}_{\omega}(s_t) \tag{2.5}$$

where k denotes the current iteration and α denotes the learning rate. This update rule is shown to converge under mild conditions if the function approximator is a linear model [116] and, in practice, often converges when using non-linear models as well (e.g. Mnih *et al.* [68, 67]). Value-based reinforcement learning methods aim to find the optimal policy implicitly by finding $Q^* := Q^{\pi^*}$. Prominent approaches include SARSA [107], which uses temporal difference learning to estimate the Q-function of the current policy and iteratively improves upon that function by defining the current policy implicitly $\pi(a|s) = \mathbb{1}(a = \text{argmax}_{a'} Q(a'|s))$. SARSA has, with the inclusion of distributed training, successfully been applied to complex domains such as Atari games [67]. Q-learning [107] uses a variation of the temporal difference learning approach to directly estimate Q^*:

$$Q^{\pi}_{\omega^{(k+1)}}(s_t, a_t) = Q^{\pi}_{\omega^{(k)}}(s_t) + \alpha \left(R(s_t, a_t) + \gamma \max_{a'} Q^{\pi}_{\omega^{(k)}}(s_{t+1}, a') - Q^{\pi}_{\omega^{(k)}}(s_t, a_t)\right) \nabla_{\omega} Q^{\pi}_{\omega}(s_t, a_t).$$

With the inclusion of replay-buffers, Q-learning was the first method to successfully use deep neural networks as a representation (DQN) to solve complex, vision-based tasks without explicitly specified features [68]. While representing the policy implicitly can lead to

instabilities during training and usually limits this approach to domains with discrete actions, Q-learning still forms the basis for state-of-the-art approaches on such domains such as Rainbow DQN [36] or R2D2 [45]. While this thesis will not use value-based methods directly, temporal difference learning will play an important role in Chapter 5 where the methods for estimating value-functions will be used to estimate other quantities. Furthermore, we will use an actor-critic method in Chapters 8 and 9.

2.2.3 Policy gradient and actor-critic methods

The second large class of reinforcement learning approaches, policy gradient methods, aims to estimate the gradient of the reinforcement-learning objective with respect to the policy parameters, i.e. to find $\nabla_\theta J(\pi_\theta)$. Many gradient estimators exist, here we will primarily review the policy gradient theorem [108] for its importance as well as for the intuition it lends to the derivations in Chapter 7, as well as deterministic policy gradients which we will use directly in Chapters 8 and 9. A third, much less well-known gradient estimator has been proposed by Morimura et al. [69]. Morimura et al. propose a generalized temporal difference learning approach to estimate the policy-gradient. We will use a similar derivation in an imitation learning context in Chapter 5 and discuss the method there in more detail.

The policy gradient theorem for the discounted-reward objective, likely the most well-known formulation of the policy gradient, can be stated as

$$\nabla_\theta J(\pi_\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \left(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)\right)]. \tag{2.6}$$

The corresponding value functions can be estimated with temporal-difference learning methods and the combined approach is being used in many state-of-the-art reinforcement learning systems such as, most recently, IMPALA [24] or Dactyl [4]. The approach is also the foundation for methods which aim to approximate the natural gradient of the objective,

e.g. [82, 98, 99].

Deterministic policy gradients [103] derive the policy gradient for deterministic policies by maximizing the Q-function at every step:

$$\nabla_\theta J(\mu_\theta) = E_{\mu_\theta}[\nabla_\theta \mu_\theta(s) \nabla_a \, Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}]. \tag{2.7}$$

Formalizing the gradient in this way has enabled the development of particularly sample-efficient systems such as D4PG [10] and TD3 [27] which achieve state-of-the-art results on certain control domains. TD3 is of particular interest to this thesis as we use it in Chapters 8 and 9. The method employs deterministic policy gradients in an off-policy setting, where the samples for the expectation come from a replay buffer. Furthermore, the method introduces a variety of heuristics to improve stability. In particular, TD3 uses 2 separate estimators for the Q function, keeps delayed target networks for policy- and value-networks and adds noise to the next-action (effectively learning the Q function of an exploration policy rather than the deterministic policy).

2.2.4 Direct search methods

Finally, black box optimizers have been used successfully to approach the reinforcement learning problem. On episodic tasks, such methods obtain sampled estimates of $J(\pi_\theta)$ directly by sampling entire episodes and computing the overall reward obtained in each episode. Direct search methods are able to find local optima of arbitrary functions based on sampled evaluations alone, without estimating the gradient of such a policy. Examples for such approaches having been successfully applied to reinforcement learning include CEM [62], CMA-ES [35], NES [125], Neuro-evolution based on genetic algorithms (e.g. Stanley and Miikkulainen [104]), REPS [81] and PI^2 [113]. While such methods have played a less prominent role in reinforcement learning research, recent results have shown that they are able to achieve comparable results to value-based or policy gradient methods

on complex benchmark tasks (e.g. Conti *et al.* [18]).

We will use REPS in Chapter 4 to find optimal trajectories to solve robotic manipulation tasks. Episodic REPS iteratively updates a search distribution over policy parameters by fitting the search distribution to its own samples, weighting each sample according to its return. The method derives the weights by deriving a closed form solution to maximizing the expected return subject to a relative entropy constraint on the change in the search distribution. In the episodic formulation, the weights are proportional to $e^{\frac{R}{\eta}}$, where R is the episodic return and η is a Lagrangian parameter, determined by optimizing the convex dual of the objective.

CHAPTER 3
IMITATION LEARNING

3.1 Problem set-up

With the foundation of sequential decision making in place, we can now our focus to the central topic of this dissertation: imitation learning. Like reinforcement learning, imitation learning is a sequential decision making problem where the agent attempts to find the optimal policy to determine its behavior. Unlike reinforcement learning, the agent does not have a reward signal which tells it exactly how good each action and each policy is (or if it does, the reward signal is not sufficient to find the optimal policy and plays an auxiliary role). Instead, an expert teacher shows the agent which behavior is desirable and the agent attempts to learn a policy which imitates the expert's behavior. Imitation provides a natural interface for the expert to communicate the desired behavior and can provide the agent with a denser signal to learn from than reinforcement learning. Naturally, it comes with its own set of challenges. For one, the imitation learning problem is much less well-defined than the reinforcement learning problem. While reinforcement learning admits different objectives (e.g. maximizing the discounted long-term reward, the average reward or the reward over finite horizon), those objectives follow a fairly similar structure. In imitation learning, the objectives are much less uniform. This is the case, even if we ignore the large variety of research questions that surround imitation learning, such as how to translate observations from the expert's point of view to the agent's point of view, i.e. the correspondence problem (e.g. [117, 101]), the best way to integrate and record data from a human teacher (e.g. [16] or whether the expert's demonstration should be treated as optimal behavior or can be improved upon (e.g. [111, 37]). Even if we focus solely on imitating the expert as accurately as possible, we can find multiple interpretations of how this is best achieved.

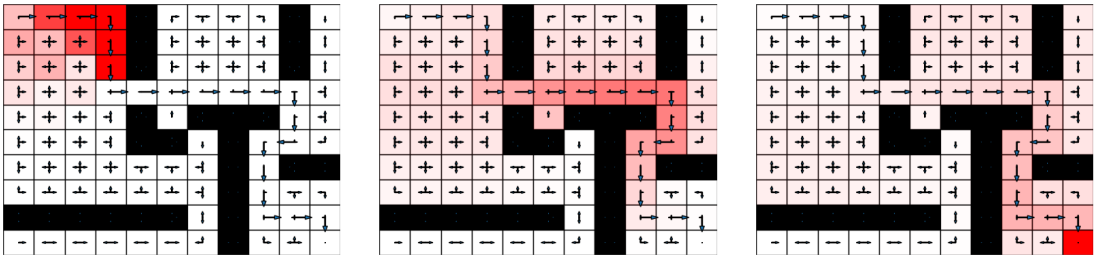
After all, the number of demonstrations is finite and the policy has to learn to generalize to unseen circumstances.

For the remainder of this chapter, we will review three different formulations of the imitation learning problem. In the first, interactive imitation learning, the expert continues to provide demonstration data while the agent is learning. While the approaches introduced in this dissertation assume that the set of demonstrations is fixed, the work in this setting has been important to frame central issues in imitation learning. In the second and third formulation, the set of demonstrations is fixed and we either have a set or a sequence of demonstrated states and actions set of demonstrated expert states $\overline{S} = (\overline{s}^{(1)}, \overline{s}^{(2)}, \dots)$ and actions $\overline{A} = (\overline{a}^{(1)}, \overline{a}^{(2)}, \dots)$. In the second formulation, we assume a set and train the agent to find general, stationary policies, while in the third, the agent is allowed to exploit time-dependent information and tries to find policies that reproduce a specific sequence. Our review will focus on methods most closely related to our work and we refer the reader to [17] for a more complete survey.

3.2 Learning from an interactive expert

The, arguably, simplest approach to imitation learning, Behavioral Cloning (BC), is to view the problem as a supervised learning problem. Given a set of example inputs, in this case $\overline{S}$, and a set of example outputs, in this case $\overline{A}$, we can use off-the-shelf supervised learning methods to learn a mapping from states to actions. This approach has been used for a long time and has led to early successes such as the first autonomous car ALVINN [84]. However, while BC is still an effective strategy and is often the easiest approach to implement in practice, it suffers from a number of shortcomings. The central problem of this approach is that it is not able to use deeper knowledge about the environment, whether it is gathered by an agent exploring the domain or simply provided by the developer. Specifically, the agent is only able to use similarity to demonstrated states to reason over which action it should take in an unseen state. This leads to two issues that are central to this thesis. First,

the agent is unable to learn from sparse data. If the agent has to follow a trajectory to solve a task, yet parts of the trajectory are missing from the demonstration data, Behavioral Cloning alone will not provide the agent with a way to learn how to complete this trajectory. Second, Behavioral Cloning suffers from the problem of accumulating errors. Whenever an agent following a learned policy makes a mistake, it is going to observe states that are different from the future states that are part of the demonstration trajectory. Without additional demonstrations, the agent will likely not be able to take the right action in this unseen state and thus make further errors. This problem has been highlighted by Ross and Bagnell [91] who note that the imitation learning set-up violates a central assumption made in supervised learning, namely that the distribution of inputs is fixed and samples from this distribution are independent. In imitation learning, the input distribution is directly affected by the prediction made by the algorithm. See Figure 3.1 for an illustration of this problem on a simple grid world domain. Ross et al. identify that the problem can be addressed by incorporating the expert in an interactive loop. By actively querying the expert on states that the agent sees during execution, interactive imitation learning methods are able to maximize agreement with the expert on states that the agent will actually see and overcome the problem of accumulating errors. One of the first approaches in this class is SEARN [19]. When applied to imitation learning, SEARN starts by following the experts action at every step, then iteratively uses the demonstrations collected during the last episode to train a new policy and collects new episodes by taking actions according to a mixture of all previously trained policies and the experts actions. Over time SEARN learns to follow its mixture of policies and stops relying on the expert to decide which actions to take. Ross et al. [91] first proved that the pure supervised approach to imitation learning can lead to the error rate growing over time. To alleviate this issue, they introduced a similar iterative algorithm called SMILe and proved that the error rate increases near linearly with respect to the time horizon. Like SEARN, SMILe iteratively queries an expert on states seen during execution of the current policy, trains a new policy using supervised learning and updates the policy



(a) After 5 steps (b) After 20 steps (c) After 40 steps

Figure 3.1: Simple grid world with a policy trained by BC. The agent is able to move in 4 directions with arrows representing the probability for an agent choosing one direction. Red shading shows density. The agent is highly certain on a trajectory but uncertain outside this trajectory. As the agent randomly steps off the path, it no longer knows what to do, errors accumulate and the agent has a non-negligible density in any state after 40 steps.

to be a mixture of all past policies. Building on this, Ross et al. introduced DAGGER [92]. DAGGER provides similar theoretical guarantees and empirically outperforms SMILe by augmenting a single training set during each iteration based on queries to the expert on the states seen during execution. DAGGER does not require previous policies to be stored in order to calculate a mixture. Note that while these algorithms are guaranteed to address the issue of straying too far from demonstrations, they require the expert to provide additional demonstrations during the agent's execution. While this approach is straight-forward to implement, stable and yields high performing policies, it is not always applicable due to this reason. Supervising a robot, for example, and recording demonstrations while the robot is learning to solve a manipulation task requires significantly more effort from the human teacher than an approach which is able to utilize a sparse set of demonstrations that have been provided up-front.

3.3 Learning from a fixed set of transitions

While interactive methods address the problem of accumulating errors effectively, the assumptions made by such methods can easily make them not applicable. It may be difficult to quickly request the expert's input during training, for example when teaching a robot

by moving it physically (kinesthetic teaching), or the expert may not be available during the entirety of training. A more general, and arguably the most common formulation of imitation learning instead assumes that the agent is trying to learn from a fixed set of demonstrations. Such demonstrations could have been recorded by the expert ahead of time or they could have been gathered by observing the expert, even without the expert's knowledge. Here, we will first look at the problem of learning stationary policies from a fixed set of transitions. This setting allows the agent to learn behaviors that generalize well to other situations and learn alternative solutions to the problem that the demonstration is solving, if necessary.

With the scenario now laid out, the problem appears to be more well-specified. However, it still admits more than one possible objective for the agent to optimize and many possible solution approaches. The disambiguity comes from the many ways in which an agent might generalize the observed behavior and reason about how to act in unseen parts of the state-space. This question always arises in a machine learning problem and does not have an objectively correct answer (no free lunch), but in imitation learning the question also involves the environment dynamics. We can generalize our behavior, even if the representations of states and actions are discrete sets and admit no reasonable similarity metric. One objective of particular interest to this dissertation is state-action distribution matching. I.e. to explicitly match the distribution of states and actions that the agent will see to that of the expert. This objective will be at the core of many approaches discussed in this section and we will introduce novel approximate maximum-likelihood methods in Chapters 5, 7 and 9. When talking about interactive imitation learning methods, we already saw the notion of matching the expert's action in states that the agent will actually see. With continuing access to the expert we can query the expert on such states, an impossibility in our chosen setting. In lieu, if we want to follow the same principle, we can ensure that the agent sees the same states as the expert and takes the same actions as the expert in those states, i.e. matching the state-action joint-distribution. Beyond this, the objective encodes a sense

of expediency. The agent will rush to visit the desired states with the right frequency which is often desired when no other objective is given. In a navigation task, simple directions at important junctions are sufficient for us to understand the desired behavior. Going quickly from junction to junction would be a reasonable policy.

To start our review, let us consider again the arguably simplest imitation learning algorithm: behavioral cloning. In the previous section, we explored the short-comings of this method to motivate interactive learning methods. It is important to note that behavioral cloning is a valid surrogate of the state-action distribution matching objective that forms the basis for many more advanced algorithms to be introduced below. As the state-distribution is unique for each given policy, if we match the policy everywhere, we will also match the state-distribution and therefore the state-action distribution. Any shortcoming behavioral cloning exhibits in contrast to other methods using the same objective thus comes from its implicit approach which ignores the distribution of states. We will argue in later chapters that explicitly matching the state-distribution yields a more robust approach to imitation learning.

The most common alternative formulation of the imitation learning problem, especially among the early work, is the inverse reinforcement learning (IRL) framework. In IRL, the agent attempts to learn a reward function from data, a premise first introduced to imitation- and reinforcement learning by Ng and Russell [73]. The promise of this approach being that the reward function is a more portable or succinct representation than the policy itself (a claim that may be true for hand-designed reward functions, but can't generally be true for all reward functions. The mapping from reward functions to optimal policies is a many-to-one mapping). Many objectives can be chosen to find such reward functions. In fact, the method proposed by Ng and Russell [73] is primarily heuristic and the desired objective is not explicitly stated. Later IRL approaches use more principled objectives to find a plausible reward function. Two common approaches can be seen in the years following the introduction of IRL. The first important idea is the idea of matching feature-expectations,

i.e. finding a policy such that the the expected mean of each feature agrees with the demonstration data [1]. Justification for this objective comes in form of a bound on the true reward the agent would accumulate. If the expert's true (unknown) reward function is linear in the given features, a bound on the deviation in feature expectations implies a bound on the difference in accumulated reward. Abbeel and Ng [1] propose two iterative algorithm to find such a linear reward function, assuming that the agent has access to the full MDP and can solve it in each iteration. However, multiple policies may lead to the same feature expectations and multiple reward functions may lead to the same policy. We can see that the objective is still underspecified. A possible solution can be found in the principle of maximum (causal) entropy: in the case of ambiguity, the agent should maximize entropy, i.e. minimize assumptions. The widely influential Maximum Entropy IRL approach has been introduced by Ziebart *et al.* [129, 128] who maximize entropy with a feature expectation matching constraint which yields a gradient-based update rule for the reward function. MaxEntIRL is still assuming a linear reward model as well as access to the MDPs, two limitations that have been challenged by follow-up work. Relative Entropy IRL (RelEntIRL) [13] propose a model-free approach to maximize the MaxEntIRL objective for linear reward models while Deep MaxEntIRL [126] shows that the gradient-based MaxEntIRL approach readily extends to non-linear, differentiable reward functions. The objective in the latter work still matches feature expectations, which, if the true reward is non-linear, is justified only empirically.

The second important idea in IRL is a margin-based objective. Here, the agent attempts to find a reward function which maximizes the difference between the expert's reward and the agent's reward (assuming the agent follows the optimal policy). Intuitively, this approach finds a reward function on which the agent performs best in the worst case. Variations of this objective have been utilized by Ratliff *et al.* [88] who use a QP-based approach to find the sub-gradient of the objective. In the max-min formulation, the objective can be phrased as a zero-sum game. Syed and Schapire [111] exploit this to find an

online-learning solution, again assuming that the reward function linear and the transition dynamics are known. Later, Syed *et al.* [110] also formulate a linear programming based solution to the same objective under slightly different assumptions. While those objectives appear to be fairly different, Ho and Ermon [39] show certain equivalencies between those formulations. For linear reward functions, matching feature expectations is shown to be equivalent to the max-min formulation of the margin objective. Furthermore, Ho and Ermon show that the adversarial objective can be seen as the dual to the objective of matching the expert's state-action distribution. If the space of reward functions is unrestricted, the objective matches the state-action joint-distribution exactly.

These early methods tended to exploit linearity of the reward function and assume a cheaply solvable MDP. Recent years have seen a number of methods that do away with these restrictions. One of the earliest methods that was successfully applied to deep neural networks is Guided Cost Learning. Finn *et al.* [25] based their method on the maximum entropy objective, adapted to a non-linear reward and therefore not matching feature expectations. Instead, the method models the distribution of trajectories $p(\tau|\overline{S}, \overline{A})$ as a maximum entropy distribution. GCL estimates the gradient of said objective based on importance-weighted samples and uses a model-based RL approach to update the sampling distribution iteratively, in parallel to learning the reward function. Following this work, Ho and Ermon [39] proposed Generative Adversarial Imitation Learning. Taking notes from Generative Adversarial Networks [30], this hugely influential method trains a discriminator to distinguish between the expert's and agent's transitions and use it as a reward function. This is shown to be a general way to match the expert's state-action distribution and has spawned a large body of work following it (e.g. [60, 65, 127]). Unlike most previous methods discussed in this chapter, GAIL is not an IRL method. While the discriminator serves as a reward function, the reward is not fixed and will be non-informative at convergence. However, adversarial training can also be used to recover a fixed reward function. Adversarial Inverse Reinforcement Learning [26] imposes a structure on the discriminator, predicting

the output probability as a function of the a valid reward. Fu et al. show that the reward approximator will be optimal in the MaxEntIRL sense. Lately, we have also seen further approaches emerge that guide the agent toward observations similar to those seen in the demonstrations, but don't try to match the distribution exactly. Random Expert Distillation [123] achieves this by learning a similarity function through support estimation. Whether or not a state is like a demonstrated state (and therefore should get a positive reward) is based on an estimator that tells the agent whether the expert's state-distribution's support extends to the state. Support of the distribution is estimated through random network distillation. Training one network to look like another via supervised-learning using demonstration data, the networks will be close in states that are similar to the one's in the demonstration and far apart otherwise. By assigning a positive reward to such states, the agent is compelled to visit those states. Assigning a reward state to demonstration states directly is impossible as the state-space is continuous. Additional measures have to be taken to incorporate this into the learning algorithm. An alternative approach, SQIL [89] assigns positive rewards to demonstration states by manipulating the replay buffer of an off-policy RL method. By ensuring that the expert transitions are going to be sampled by the RL method, the method can heuristically assign positive reward to expert transitions.

3.4 Learning from time-indexed sequences

One thing all afore-mentioned methods have in common is that they are trying to learn a stationary policy that generalizes to new situations. Often, however, we are satisfied if the agent is able to reproduce a specific sequence in a robust way. For example, we may wish a robot to perform a specific manipulation task repeatedly and reliably. Another reason might be a hierarchical approach, which generalizes to arbitrary situations but utilizes specific, learned sequences in order to do so, e.g. Kulić *et al.* [53], Mülling *et al.* [70], and Merel *et al.* [64]. Of course, the general methods we reviewed in the previous section still apply, but it is possible to derive more specialized methods that take the sequential nature of the

demonstration data into account. Here, we briefly review a variety of ways such temporal information can be incorporated into imitation learning. Primarily, this serves to give a more complete picture of the field, but we also utilize a trajectory-based method in Chapter 4.

If we are willing to decouple the control problem from the imitation learning problem, we can use supervised learning to learn the trajectory that the agent should reproduce. Any representation for sequence data can be used as a trajectory representation (e.g. Kulić *et al.* [53]), but arguably the most influential representation in the imitation learning context are Dynamic Movement Primitives (DMPs) [41]. DMPs represent the trajectory as a dynamical system, drawing the agent to a specific goal-position while super-imposing a shaping function. We will explain DMPs in more detail in Chapter 4, where we use them extensively. A relevant variation on the concept are Probabilistic Movement Primitives [77], which explicitly incorporate noise in the description of the shaping term in order to define a number of probabilistic operations that work with the movement primitives.

Trajectory representations such as DMPs allow us to learn the trajectory and generalize in the state-space, but do not teach the robot directly how to act. A separate controller must be given or learned in order to follow the desired trajectory. Recently, a number of approaches have been proposed that are able to efficiently utilize time-indexed demonstration data to learn a policy end-to-end. One approach is to define a domain-specific, time-dependent (or otherwise non-stationary) reward-function that enables the agent to learn to track a desired trajectory. This approach can be quite powerful, if such a reward-function can be specified. One example of this is the work by Aytar *et al.* [8], who train an agent to play Montezuma's revenge from video by extracting a series of checkpoints. Whenever the agent reaches a checkpoint for the first time, it is given a positive reward. Other examples for this include Merel *et al.* [63] and Peng *et al.* [80]. Both works specify task specific reward functions that enable a humanoid agent to track movements of a motion capture recording in simulation.

Pathak *et al.* [78], instead of teaching the agent to reproduce a specific trajectory, train a goal-conditioned skill-policy that takes in the next-state the agent should reach. To track a given demonstration then requires no further training-time. The agent can follow a desired trajectory by sequentially conditioning on the next reference state while interacting with the environment.

Finally, it is also possible to follow a hybrid approach. Wang *et al.* [124] and Duan *et al.* [22] both train a general policy, but condition it on an encoding of the entire trajectory that they wish it to follow. Duan et al. use an LSTM to produce the encoding and apply this approach to the meta-learning setting. Using the encoding, the agent is able to learn to generalize more quickly to new trajectories. In contrast, Wang et al. use this approach to produce more diverse trajectories. The encoding is learned using a variational auto-encoder with a Wavenet decoder, allowing the agent to not only encode given trajectories but also sample valid encodings of other trajectories that match the demonstrations. By providing the agent with such an encoding, the policy, trained using GAIL, no longer has to capture diversity of different solutions.

CHAPTER 4
GUIDING POLICY SEARCH WITH VIA-POINTS AND DYNAMIC MOVEMENT PRIMITIVES

4.1 Motivation

In the previous chapter, we briefly reviewed many of the great advances that have been made in the field of imitation learning; however, utilizing extremely sparse imitation data remains a difficult challenge. For the remainder of this dissertation, we will introduce a series of imitation learning algorithms that address these difficulties, the first of which will be an algorithm that is able to utilize sparse imitation data and is specifically designed for trajectory learning for robot manipulation tasks. The algorithm has first been introduced in Schroecker *et al.* [95]. As in the remainder of this thesis, we assume the model of the environment is unknown and that the robot must learn how to act.

Learning to perform difficult robot manipulation tasks is one of the central problems in the field of robotics. Both, reinforcement- and imitation learning have shown great promise for autonomous learning of such motor skills (see Chapters 2 and 3). Both, however, still have many shortcomings that need to be addressed to fully solve the challenges that arise in this domain. Reinforcement learning in complex and high dimensional spaces is often still limited by the necessary amount of exploration, especially when the number of interactions with the environment is limited, as is often the case in robotics. Imitation learning, on the other hand, can avoid this issue when it is used to teach correct behavior directly or to drastically reduce the sample-complexity of a reinforcement-learning method (see e.g. Kormushev *et al.* [51], Mülling *et al.* [70], and Chernova and Thomaz [17] and Chapter 3); however, complete demonstration trajectories can be difficult to record on robot manipulation tasks. First, providing complete demonstrations requires the user to have a full under-

standing of the task. In many situations this requirement is too restrictive as the user may only have partial information about important milestones of the task. Second, the user's ability to provide demonstrations is limited by the available input modalities. Teaching complete trajectories using tele-operation or kinesthetic demonstration can lead to undesirable pauses, sprints and imperfections. Akgun *et al.* [2] show an effective alternative to recording full trajectories by using a sequence of individually recorded poses as demonstrations. The authors show the approach to lead to smoother trajectories while being easier to record. We refer head to Figures 4.1b and 4.2a for examples of such demonstrations on a robotic manipulation and a synthetic letter writing task respectively.

This leaves the question of how the agent should behave between demonstrated via-points. Existing work often solves this problem heuristically, e.g. via task-space interpolation. We will review such methods in Section 4.2.1. No solution can exist that will always lead to correct behavior, but different approaches can be derived based on assumptions made. In this chapter, we will assume a task-oriented set-up and assume the existence of a reward function that can teach the robot how to act within the constraints of the given demonstrations. We will show that this enables the agent to utilize partial trajectories while simultaneously being able to learn the shape of the trajectory beyond simple interpolation. In the following chapters, we will explore a pure imitation approach that disambiguates between possible solutions based on a distributional measure.

Specifically, in this chapter we derive an approach which learns a trajectory, represented by a Dynamic Movement Primitive (DMP) [42], using direct policy search algorithms such as as PoWER [50], REPS [81], policy search based on CMA-ES [35, 105] or PI^2 [112]. We propose to record soft via-points which the robot has to pass through (within some margin of error), to be used as demonstrations. The search space of the policy search method is then constrained by those via-points. The key hypothesis of this approach is that reasoning over desired future states, as encoded in the DMP, is sufficient when the learner is additionally provided with a reinforcement learning signal guiding it toward the correct

solution and is able to override the demonstration signal to the extent necessary to adjust for discrepancies between planned future states and observed future states. For example, while the trajectory represented by a given DMP may be blocked by an obstacle and thus the observed trajectory may not reach the demonstrated via-point, we assume that the policy search will be able to filter out such trajectories implicitly. We will relax this assumption in the coming chapters.

Besides validating the central thesis by showing that such a via-point approach is effective and can learn from only minimal demonstrations, we also explore another advantage of this method. Utilizing via-points in this way enables the expert to correct the agent during the learning process. Typically, demonstrations are provided as one, in the beginning of the learning process. If we are not satisfied with the behavior that the agent has learned, either via imitation or later via reinforcement learning, the only recourse is to re-record entire demonstrations and restart the learning process. Refining a learned policy and removing undesirable effects is usually difficult and cannot isolate specific aspects of the learned trajectories. Here, however, as our method treats via-points as probabilistic constraints, we can add via-points later on during the learning process. We show that this enables the expert to provide interactive corrections even after the reinforcement learning algorithm has started to improve on the agent's policy and the expert had a chance to observe the agent's behavior.

4.2 Background

4.2.1 Via-points as demonstrations and interactive corrections

Teaching via-points by demonstrations is also known as key-frame demonstration and has been shown to constitute a user-friendly and efficient way to obtain demonstrations [3]. Wada *et al.* [122] extract via-points from a continuous demonstration and show that these can be used to create a trajectory that minimizes torque change. Miyamoto *et al.* [66] extend this approach and apply it to learning robot motor skills. Bitzer *et al.* [12] have presented an

approach that combines key-frame demonstrations with reinforcement learning by learning a lower-dimensional manifold to simplify the state-space for a non-episodic reinforcement learner. This method differs from our approach in that it only learns a simpler state-space representation and does not learn a heuristic for specific trajectories.

In this chapter, we propose to probabilistically constrain the search space over DMP parameters and find trajectories that pass through the desired via-points. Another representation of distributions over trajectories that can be restricted to go through specified via points is called Probabilistic Movement Primitives and has been proposed by Paraschos *et al.* [77]. The conditioning operation defined in this approach is similar to the operation for distributions over DMPs introduced in Section 4.3.2. However, while probabilistic movement primitives could also be used as a parametric policy for our approach, we are focusing on Dynamic Movement Primitives as they are more popular and better understood.

Finally, we propose to utilize via-point demonstrations to enable the teacher to provide interactive corrections during the learning process.

Utilizing corrections in order to change a policy learned from demonstration has been previously introduced by Argall *et al.* [6] who propose to use tactile corrections. While this approach is interesting, it cannot be straight-forwardly integrated into autonomous reinforcement learning algorithms such as the ones utilized in our approach.

Interactive reinforcement learning utilizes feedback from human teachers to guide the learning process. Methods such as introduced by Knox and Stone [48], Judah *et al.* [44] or Griffith *et al.* [33] use reward-like signals or action advice obtained by a human teacher to shape the reinforcement learning process and to allow the expert to correct the behavior of the agent during the learning process by modifying the reward- or value-function or modifying the behavior policy locally. Compared to interactive reinforcement learning methods, our approach utilizes corrections more directly. Using a trajectory-based representation, we can prune the search space, allowing our corrections to have a more immediate effect.

4.2.2 Direct Policy Search

To find the parameters for a parametric trajectory representation within the constraints of the given via-points, we propose to utilize methods such as PoWER [50], which obtains a policy as a weighted regression of the old samples, REPS [81], which derives a similar update rule from an information theoretic formulation of the learning problem, CMA-ES [35], a black-box optimizer that has been used for policy search [105] or PI^2 [112], which uses a method based on path integrals to improve the policy iteratively. Let π_θ be the parametric policy, these methods maintain a Gaussian search distribution $p^\pi_\omega(\theta)$, where ω refers to the parameters of said policy distribution. During the learning process, the search distribution is iteratively updated using the cumulative reward of entire roll-outs.

For our evaluation, we propose to use Relative Entropy Policy Search (REPS) [81] as the search method. REPS attempts to find the optimal policy subject to a constraint which limits the kl-divergence between the current search distribution and the distribution in the following iteration. I.e. in our case, it repeatedly finds an approximate solution to the following optimization problem:

$$\begin{aligned} &\max_\omega \int p^\pi_\omega(\theta) E_{\pi_\theta}[R]\, d\theta \\ &s.t. \epsilon \geq \int p^\pi_\omega(\theta) \log \frac{p^\pi_\omega(\theta)}{p^\pi_{\overline{\omega}}(\theta)} d\theta \end{aligned} \tag{4.1}$$

where θ are the parameters of the DMP, $E_{\pi_\theta}[R]$ is the expected total return when following a given DMP, ω are the parameters of a Gaussian search distribution $\overline{\omega}$ are the parameters of the previous iteration, and ϵ is a bound on the divergence to the previous iteration, serving as a step size. Peters et al. find a closed form, sample-based solution by sampling DMPs from $p^\pi_{\overline{\omega}}$, calculating weights and fitting ω to the weighted samples, i.e.

$$p^\pi_\omega(\theta) \propto \int p^\pi_{\overline{\omega}}(\theta) e^{\eta^{-1} E[R|\theta]} d\theta. \tag{4.2}$$

Here, η is a Lagrangian parameter that follows from minimizing the convex dual

$$\min_{\eta} \eta \log \int p_{\bar{\omega}}^{\pi}(\theta) e^{\eta} d\theta \tag{4.3}$$

REPS belongs to the family of direct search methods as it maintains a search distribution which is iteratively updated, using a closed-form and gradient free approach. In Section 4.3, we will discuss how to modify the search distribution directly in order to incorporate via-point demonstrations.

4.2.3 Dynamic Movement Primitives

Policy search relies on optimizing the parameters of a parametric policy representation. One such policy representation are Dynamic Movement Primitives which have been introduced by Ijspeert et al. in [42]. DMPs are defined as dynamical systems that are attracted by a goal position while following a superimposed trajectory. As such they have the property that they can adjust the goal position of the trajectory separately from the shape of the trajectory. Furthermore, DMPs incorporate the concept of a *phase*, an abstraction of time allowing for easy adjustment of the overall time scale. DMPs are defined as

$$\ddot{y} = \tau^2 \alpha \left(\beta(g - y) - \frac{\dot{y}}{\tau} \right) + \tau^2 f_{\mathbf{w}}(z), \tag{4.4}$$

$$\dot{z} = -\tau \alpha_z z. \tag{4.5}$$

where $y \in \mathbb{R}^N$ defines the N dimensional state (e.g. joint position), $\tau \in \mathbb{R}$ is a time scaling parameter and $\alpha_z \in \mathbb{R}$ is a parameter that shapes the phase function. $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$ are parameters that are analogous to the gains of a PD-controller and define how the system is drawn to the goal of the trajectory which is defined by $g \in \mathbb{R}^N$. $f_{\mathbf{w}} : \mathbb{R} \to \mathbb{R}^N$ is the forcing function which determines the shape of the trajectory and is defined as a mixture of radial

basis functions with parameters K, c_i and h_i for $0 \leq i < K$.

$$f_{\mathbf{w}}(z) = \frac{\sum_{i=1}^{K} \varphi_i(z) w_i}{\sum_{i=1}^{K} \varphi_i(z)} z, \qquad \varphi_i(z) = \exp\left(-\frac{1}{2}\frac{(z-c_i)^2}{h_i}\right). \tag{4.6}$$

Commonly, the weights $w_i; 0 \leq i < K$ as well as the goal position are taken as the parameters $\theta = \binom{g}{\boldsymbol{w}}$ of the DMP that are learned by demonstration or autonomously whereas the other parameters are given as hyper-parameters. K then determines the dimensionality of the parameter-space. In this work, we are operating on Gaussian search distributions over DMP parameters to find the desired trajectory. To generalize over multiple instances of the task, we will also consider the situation where the mean of this search distribution is a linear function of context parameters $\mathbf{\Phi}$, i.e. $p_\omega^\pi = \mathcal{N}(\cdot | A_\pi \mathbf{\Phi}, \Sigma_\pi)$.

4.3 Approach

4.3.1 Guiding policy search

Relative Entropy Policy Search or other direct search algorithms allow us to iteratively update a search distribution in the space of DMPs to find a near-optimal trajectory. Partial demonstrations in the form of via-points allow us to incorporate additional constraints to drastically reduce the number of roll-outs required to find a good trajectory. To define this process in more specific terms: we propose a setup where the teacher is observing the policy search learning process and can, before starting the learning process (demonstrations) or in-between iterations (corrections), inspect trajectories sampled from the current search distribution of the probabilistic policy, provide suggestions by physically or remotely moving the robot and then accept or reject the modified distribution. Suggestions are recorded as soft via-points y^* that the trajectories have to pass through at a specified time t^*. The time t^* associated with the via-point can either be recorded as well, for example if the user is stopping the robot during an execution in order to provide a new via-point, or can be estimated manually based on domain knowledge. In Section 4.4, we treat the given via-points

as evenly spaced. As a parametric trajectory representation with limited capacity, it may not be possible for a DMP to comply with the given via-points exactly. Furthermore, as DMPs specify the desired trajectory rather than the actual trajectory that is subject to environment dynamics, small deviations from the demonstrations in the idealized trajectory may lead to better performance in actuality, even if the provide demonstrations are optimal. Taking this into account, we treat the provided via-points as soft constraints, allowing the robot to violate the demonstration to an extent regulated by a hyper-parameter σ_y.

The abstract form of the proposed algorithm is thus to initialize the search distribution randomly and to iterate:

1. If requested by the expert teacher, record via-point demonstrations y^*, t^*
2. Update search distribution based on via-points
3. Collect roll-outs based on DMP sampled from search distribution
4. Update search distribution based on observed rewards using a direct search method

In many cases it can be desirable to perform policy search on parameterized tasks as this allows us to learn variations of a task instead of optimizing for a single trajectory. For example, one might want to learn a policy that depends on the location and orientation of key objects instead of learning a new policy for each variation of the task. Tasks such as these can be learned by learning a policy whose mean is a linear function of some features $\mathbf{\Phi}$ of the task parameters [56]. In order to deal with parameterizable tasks, we assume that the via-points are given by a model that is linear in the features of the task parameters, i.e. $y^* = B\mathbf{\Phi}$. Such a model can, for example, be learned by using linear regression or can be specified manually based on domain knowledge. Note that for fixed via-points, one might simply choose $\mathbf{\Phi}$ to be constant.

With the algorithm now defined in the abstract, we will now consider the question of how to incorporate via-point demonstrations as soft constraints to update the search distribution.

Algorithm 1 Guiding Policy Search with Interactive Via-Points

Initialize $p^{\pi^{(0)}}(\boldsymbol{\theta}) \leftarrow \mathcal{N}\left(\boldsymbol{\theta}; \mu_{\pi^{(0)}}, \Sigma_{\pi^{(0)}}\right)$
Obtain initial via-points $\mathbf{y}^*, \mathbf{t}^*$ from demonstration
for $y^*, t^* \leftarrow \mathbf{y}^*, \mathbf{t}^*$ **do**
    $p^{\pi^{(0)}}(\boldsymbol{\theta}) \leftarrow p(\boldsymbol{\theta}|y^*, t^*, \mu_{\pi^{(0)}}, \Sigma_{\pi^{(0)}})$ (see Eq. 4.20)
for $k \leftarrow 1$ to N iterations **do**
    $p^{\pi^{(k)}}(\boldsymbol{\theta}) \leftarrow$ POLICY_SEARCH_ITERATION($p^{\pi^{(k-1)}}$)
    Execute trajectory defined by mean parameters $\mu_{\pi^{(k)}}$
    while teacher stops execution **do**
        Record stop time t^*
        Let teacher move the robot, obtain correction y^*
        $p^{\pi^{(k)}}(\boldsymbol{\theta}) \leftarrow p(\boldsymbol{\theta}|y^*, t^*, \mu_{\pi^{(k)}}, \Sigma_{\pi^{(k)}})$
        Execute mean trajectory defined by $\mu_{\pi^{(k)}}$

4.3.2 Updating the search distribution

Assuming that we have a via-point as obtained in Section 4.3.1, we derive a modified search distribution for the underlying policy search that passes close to this via-point at the specified time t^*:

$$p_H = p(\theta|t^*, y^*) \propto \mu(y^*|t^*, \sigma_y \boldsymbol{I}, \theta) p^{\pi}(\theta|t^*). \tag{4.7}$$

The latter distribution p^{π} denotes the prior of where we assume that the human would want the samples to lie. Here, we use the current policy $\mathcal{N}(\theta|A_{\pi^{(k)}}\boldsymbol{\Phi}, \Sigma_{\pi^{(k)}})$ as a prior. Using the current policy as a prior means that agent will aim to follow the current best estimate according the policy search unless told otherwise by the expert. Note that this prior means that we are assuming that the corrections are plausible according to the current search distribution. A different prior would be necessary, if we wish to allow the expert to

influence the agent when it has already converged to a (sub-optimal) solution.

The likelihood distribution μ denotes the probability of a given DMP going through the specified via point with a specified variance σ_y. This distribution is dependent on the trajectory that the DMP is following. As DMPs are defining accelerations as a linear differential equation, they can be solved for y given a starting position and velocity in order to obtain an estimate of the position at any given time. Note that the solution will not be exact as a real robotic system always has noise and the DMP will react to that noise as well as inaccuracies in the underlying controller. Only the goal position is being tracked exactly. However, using this estimate is reasonable as long as the noise of the estimate is significantly smaller than the inaccuracies introduced by the human teacher when recording a demonstration. Assuming that we start from a rest position where position y_0 and velocity $\dot{y}_0$ is zero (we make this assumption for the sake of simplicity and will relax it in the following section), the dynamical system can be solved for the position y using Duhamel's principle:

$$\begin{pmatrix} \boldsymbol{y} \\ \dot{\boldsymbol{y}} \end{pmatrix} = \int_0^t h_t(s) \begin{pmatrix} 0 \\ \tau^2(\alpha\beta g + \boldsymbol{\Psi}(z(s))^T \boldsymbol{w}) \end{pmatrix} ds, \tag{4.8}$$

where

$$h_t(s) = e^{(t-s)\begin{pmatrix} 0 & 1 \\ -\tau^2\alpha\beta & -\tau\alpha \end{pmatrix}}, \qquad z(t) = e^{-\tau\alpha_z t}, \qquad \boldsymbol{\Psi}_i(z) = \frac{\varphi_i(z) z}{\sum_{j=0}^{K} \varphi_j(z)}. \tag{4.9}$$

This equation is linear w.r.t the parameters θ and can therefore be written as:

$$\boldsymbol{y} = \begin{pmatrix} 1 & 0 \end{pmatrix} \left(\int_0^t h_t(s) \begin{pmatrix} 0 \\ \tau^2(\alpha\beta g + \boldsymbol{\Psi}(z(s))^T \boldsymbol{w}) \end{pmatrix} ds \right) \tag{4.10}$$

$$= \begin{pmatrix} 1 & 0 \end{pmatrix} \left(g\alpha\beta \int_0^t h_t(s) \begin{pmatrix} 0 \\ \tau^2 \end{pmatrix} ds \; + \; \sum_{i=0}^{N} w_i \int_0^t \Psi_i(z(s)) h_t(s) \begin{pmatrix} 0 \\ \tau^2 \end{pmatrix} ds \right) \tag{4.11}$$

$$= \boldsymbol{M}_t \theta, \tag{4.12}$$

where

$$\boldsymbol{M}_t = \begin{pmatrix} m_0 & m_1 & \cdots & m_{N+1} \end{pmatrix}, \tag{4.13}$$

$$m_0 = \int_0^t \alpha\beta h_t(s) \begin{pmatrix} 0 \\ \tau^2 \end{pmatrix} ds, \tag{4.14}$$

$$m_i = \int_0^t \Psi_{i-1}(z(s)) h_t(s) \begin{pmatrix} 0 \\ \tau^2 \end{pmatrix} ds; 0 < i < N+1. \tag{4.15}$$

Note that this equation is for a single dimension but can straightforwardly be extended to multiple dimensions by extending $\boldsymbol{M}$ diagonally and adding rows to θ such that

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{M}_t & \mathbf{0} & \cdots \\ \mathbf{0} & \boldsymbol{M}_t & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} \boldsymbol{\theta_0} \\ \boldsymbol{\theta_1} \\ \vdots \end{pmatrix},$$

where θ_j denotes the parameters of the DMP for dimension j. Therefore for DMPs, we can write the likelihood distribution μ as

$$\mu(y^*|t^*, \sigma_y \boldsymbol{I}, \theta) = \mathcal{N}(y^*|\boldsymbol{M}_{t^*}\theta, \sigma_y \boldsymbol{I}). \tag{4.16}$$

Now we can calculate the sampling distribution as the posterior distribution of the likelihood μ, encoding information about the via-point, and the prior distribution p^π which consists of the previously learned policy. For Gaussian distributions the sampling distribution is therefore given as

$$p(\theta|t^*, y^*) = \mathcal{N}(\theta|A_H \boldsymbol{\Phi}, \Sigma_H), \tag{4.17}$$

$$\Sigma_H = \left(\Sigma^{-1}_{\pi(k)} + \boldsymbol{M}^T_{t^*} \sigma_y^{-1} \boldsymbol{I} \boldsymbol{M}_{t^*}\right) \tag{4.18}$$

$$= \Sigma_{\pi(k)} - \Sigma_{\pi(k)} \boldsymbol{M}^T_{t^*} \left(\sigma_y \boldsymbol{I} + \boldsymbol{M}_{t^*} \Sigma_{\pi(k)} \boldsymbol{M}^T_{t^*}\right)^{-1} \boldsymbol{M}_{t^*} \Sigma_{\pi(k)}, \tag{4.19}$$

$$A_H = \Sigma_H \left(\boldsymbol{M}^T_{t^*} \sigma_y^{-1} \boldsymbol{I} B + \Sigma^{-1}_{\pi(k)} A_{\pi(k)}\right). \tag{4.20}$$

Where the application of the Woodbury matrix identity in Eq. 4.19 allows for numerically stable computation of that distribution.

4.3.3 Deriving a sampling distribution for arbitrary starting positions and velocities

In the previous section, we derived a sampling distribution under the assumption that $y_0 = 0$ and $\dot{y}_0 = 0$. However, while $y_0 = 0$ can be assumed w.l.o.g., $\dot{y}_0 = 0$ is only true for trajectories that start from a rest position. Here, we derive an equivalent sampling distribution for the general case. In the general case, the closed form for dynamic movement primitives is given by

$$\begin{pmatrix} \boldsymbol{y} \\ \dot{\boldsymbol{y}} \end{pmatrix} = \int_0^t e^{(t-s)\begin{pmatrix} 0 & 1 \\ -\tau^2\alpha\beta & -\tau\alpha \end{pmatrix}} \begin{pmatrix} 0 \\ \tau^2(\alpha\beta g + \boldsymbol{\Psi}(z(s))^T \boldsymbol{w}) \end{pmatrix} ds + e^{t\begin{pmatrix} 0 & 1 \\ -\tau^2\alpha\beta & -\tau\alpha \end{pmatrix}} \begin{pmatrix} y_0 \\ \dot{y}_0 \end{pmatrix}. \tag{4.21}$$

And therefore

$$\boldsymbol{y} = \boldsymbol{M}_t \theta + c \tag{4.22}$$

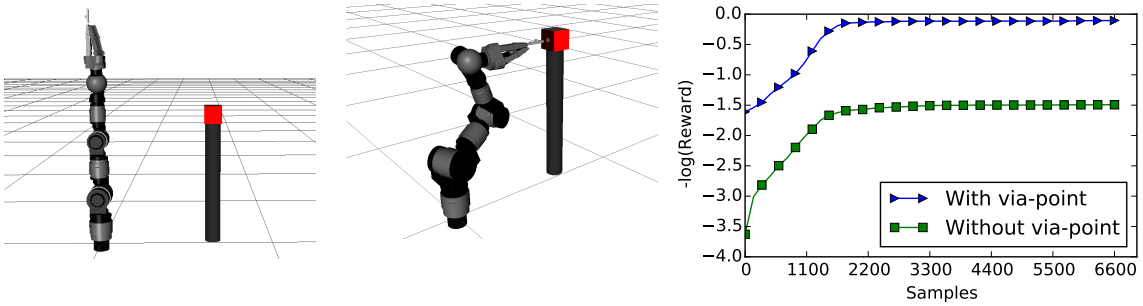
where

$$c = \begin{pmatrix} 1 & 0 \end{pmatrix} e^{t\begin{pmatrix} 0 & 1 \\ -\tau^2\alpha\beta & -\tau\alpha \end{pmatrix}} \begin{pmatrix} y_0 \\ \dot{y}_0 \end{pmatrix}. \tag{4.23}$$

The sampling distribution $p(\theta|t^*, y^*) = \mathcal{N}(\theta|A_H\boldsymbol{\Phi}, \Sigma_H)$ is then computed with the modified likelihood distribution $p(y^*|t^*, \theta) = \mathcal{N}(y^*|\boldsymbol{M}_{t^*}\theta + c, \sigma_y \boldsymbol{I})$. The mean of this distribution differs slightly from the distribution derived in the previous section so that

$$A_H\boldsymbol{\Phi} = A_H = \Sigma_H \left(\boldsymbol{M}_{t^*}^T \sigma_y^{-1} \boldsymbol{I} \left(B\boldsymbol{\Phi} - c \right) + \Sigma_{\pi(k)}^{-1} A_{\pi(k)} \boldsymbol{\Phi} \right). \tag{4.24}$$

Note that we can assume w.l.o.g. that $\boldsymbol{\Phi}$ is of the form $\boldsymbol{\Phi} = \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \vdots \\ 1 \end{pmatrix}$. The sampling



(a) (b) (c)

Figure 4.1: **a)** Rest position of the robot. The robot has to find a trajectory that puts the object in the box. **b)** Via-point with the key placed in front of the hole, as it was given to the robot. All trajectories have to be close to this easy-to-provide via-point which gives significant aid for finding the opening in the box. **c)** Rewards obtained over 20 trials during the learning process with and without via-points. In 18 out of 20 trials, the learner with the via-point finds the opening in the box and obtains a reward close to zero. Without the via-point the learning process converges to a local optimum.

distribution is then defined by

$$\Sigma_H = \Sigma_{\pi(k)} - \Sigma_{\pi(k)} \boldsymbol{M}_{t^*}^T \left(\sigma_y \boldsymbol{I} + \boldsymbol{M}_{t^*} \Sigma_{\pi(k)} \boldsymbol{M}_{t^*}^T\right)^{-1} \boldsymbol{M}_{t^*} \Sigma_{\pi(k)}, \tag{4.25}$$

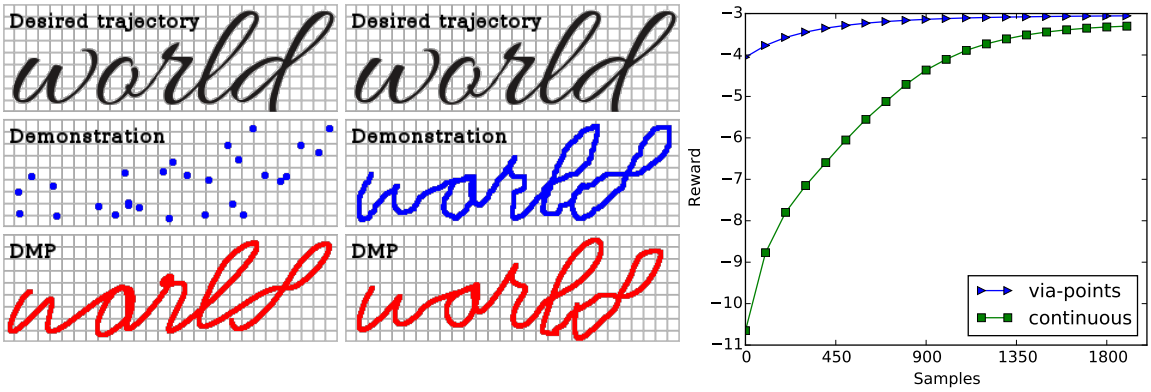
$$A_H = \Sigma_H \left(\boldsymbol{M}_{t^*}^T \sigma_y^{-1} \boldsymbol{I} \left(B - \begin{pmatrix} 0 & \dots & 0 & c \end{pmatrix}\right) + \Sigma_{\pi(k)}^{-1} A_{\pi(k)}\right). \tag{4.26}$$

4.4 Evaluation

To evaluate our method, we first utilize a simulated robot arm to show that such a modified sampling distribution can drastically improve the outcome of a reinforcement learner by having it learn how to insert an object in a hole (this is commonly known as the peg in hole problem). We then utilize a word reproduction task in order to provide a comparison to continuous demonstrations and to exhaustively analyze the key-properties of our algorithm.

4.4.1 Peg in hole

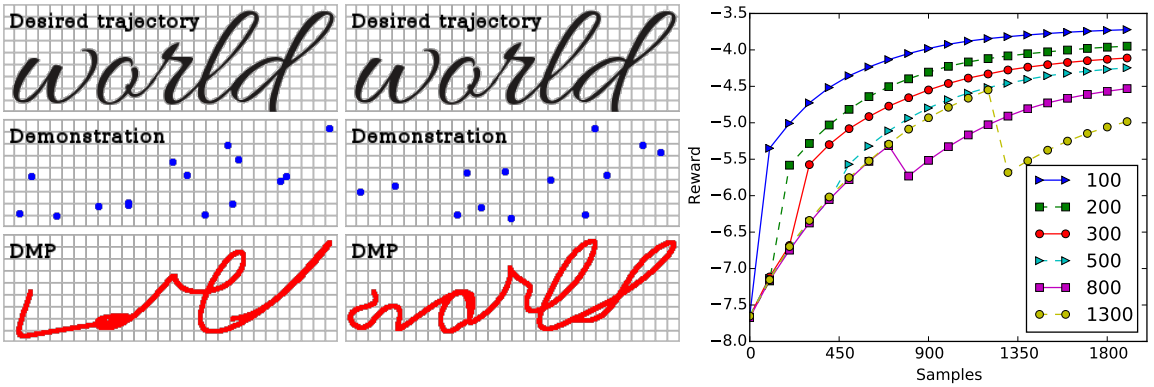
In our first experiment, we are evaluating the influence of a small number of demonstrated via-points on the reinforcement learning process and show that it can drastically improve the convergence of the policy search. To this end, we are utilizing a simulated 7 DoF robot



(a) (b) (c)

Figure 4.2: **a)** Mean trajectory of initial initial policy (bottom) derived from via-points (center). The trajectory is smooth and barely deviates from the desired trajectory (top) even in-between via-points. **b)** Mean trajectory (bottom) of initial policy derived from a continuous demonstration (center). Especially the last letter shows how the trajectory mimics imperfections of the demonstration that are amplified in time. **c)** Comparison of reward obtained within 2000 samples, averaged over 30 trials. Learning initialized with via-points consistently outperforms the learning process initialized with a continuous demonstration.

arm and have it learn how to insert an object into a hole without any knowledge about the environment. The objective is for the agent to minimize the squared distance of the tip of the object to the center of the box. Using a limited amount of samples, this is difficult for reinforcement learning as the agent has to find the opening of the box and learn the motion that allows it to insert the object while avoiding local minima around obstacles. We provide a single via-point (see Figure 4.1a) and show that this is sufficient to guide the learning process to the right solution. To learn this task, we utilize Relative Entropy Policy Search as the underlying policy search algorithm. The trajectories are represented by 7 Dynamic Movement Primitives with 6-dimensional weights leading to a 49-dimensional action vector θ. The measured reward is averaged over 20 trials with 45 policy search iterations per trial and 150 samples for each iterations. The updates are relatively aggressive to allow for the robot to learn using relatively few sample roll-outs (the process can be considered to have converged in less than 2000 roll-outs). As can be seen in Figure 4.1c, the reward curve observed by using the via-point is converging to a value close to 0, meaning that the total squared distance to the final position is converging to 0. The learning process



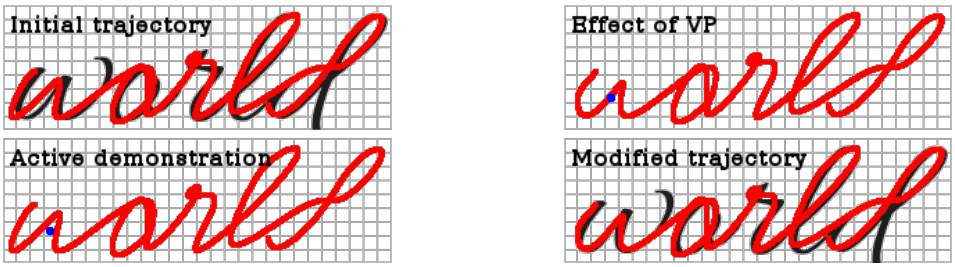
(a) (b) (c)

Figure 4.3: **a)** Mean trajectory of the initial policy (bottom) derived by using only every second via-point (center). This trajectory omits whole letters when compared to the desired trajectory (top). **b)** Trajectory after 3 iterations (bottom), when the second half of the via-points have been added (center). The "r" is still ignored because the prior is too strong. **c)** Comparison of rewards obtained with the rest of the via-points added after different numbers of samples. Early demonstrations are clearly better than late demonstrations but late demonstrations can still have a positive effect.

initialized with a random policy, on the other hand, is converging to a local minimum. What the agent has learned in this case is to move the end-effector close to the box, i.e. the robot is converging to solutions where the end-effector is pressing against the middle of other sides of the box in order to get closer to the goal position. As a consequence, it stops exploring and does not find the opening. Note that the learning process always exceeds the initial reward obtained by our approach as the reinforcement learner always finds at least the closest points outside of the box which cannot be derived by the provided via-point alone.

4.4.2 Letter writing

To further investigate the properties of this algorithm we are evaluating our approach on a letter reproduction task as well. Variations of this task have been used in the past to evaluate different properties of Dynamic Movement Primitives as it has many similarities with learning trajectories for robot arms while allowing for intuitive visualization, easy recording of demonstrations and fast execution [42, 41]. The objective of the letter reproduction



(a) (b)

Figure 4.4: Example of giving via-points interactively. **a)** Mean of original policy in comparison to the desired trajectory (top) and the via-point (bottom). **b)** Mean of the modified policy. The trajectory goes through the via-point (top) and thereby matches the desired trajectory more closely (bottom).

task is to learn trajectories for both dimensions which, when followed by a simulated pen, can accurately reproduce a sequence of letters. We are representing those trajectories as DMPs with a 60 dimensional weight-vector for each dimension and initialize the policy by a continuous demonstration or via-points before optimizing it using REPS. For the policy search, we defined the reward function as the number of overlapping black pixels: $-\frac{10^4}{\#IntersectingPixels+1}$

Comparison to continuous demonstrations

The first instance of the letter reproduction task compares learning initialized with a normal distribution around a continuous demonstration to learning initialized on a fixed set of via-points at equi-distant points in time. In this experiment we show that despite the lack of information between via-points, our approach will converge to a more accurate solution in less time. To obtain an initial policy from a continuous demonstration we use standard least squares to learn the mean and then add an initial variance of 10^3 for the weights and 5 for the goals of the DMPs. We have found these values to yield the best result in the continuous case. For policy search initialized with via-points, we start with a multivariate normal distribution with mean 0 and a variance of 10^5 for the weights and 500 for the goal positions. Note that the higher initial variance is necessary as conditioning on the via-points will otherwise lead to an overly narrow distribution. This initial distribution

is then used as the prior distribution to obtain a modified initial policy based on the via-points that can be seen in Figure 4.2a. In Figures 4.2a and b, it can be seen that the initial policy derived from the via-points is very smooth whereas the initial policy derived from a continuous demonstration mirrors the imperfections of that demonstration. Note that these imperfections are often much larger in practice and that policy search is unnecessary in domains where given demonstrations already solve the task perfectly. Furthermore, the figures show that the errors introduced by missing information between the via-points is of the same order as the errors that can be introduced by linear regression and that the mean of the initial policy is already describing a good trajectory which leads to a fast learning process. Finally, Figure 4.2c shows that our approach yields both, higher initial reward as well as higher final reward and therefore better trajectories before and after the learning process, when compared to the baseline. Note that the higher initial reward is tied to the amount of exploration that is necessary in the beginning and can, in many cases, be a desirable property when it comes to safe exploration. While our approach only explores the areas in-between the via-points, a reinforcement learner that has been initialized with a continuous demonstration has to explore around the full trajectory. It is possible to reduce the exploration around the continuous demonstrations; however this would also reduce the final reward that can be obtained.

Active corrections

One important aspect of the approach presented in this chapter is that the via-points do not need to be obtained at the start of the learning process but can be given at a later point and be utilized by an already trained policy. This is useful in situations where the optimal trajectory is not immediately apparent to the user. To investigate the impact of providing via-points at later iterations we are looking at a variation of the same learning task where only every other of the initial via-points are given before the reinforcement learning process is started. Figure 4.3a shows that this constitutes a far worse initial policy

as it omits critical parts of the trajectory completely and we would therefore expect to see great benefit from adding the second half of the suggestions. As can be seen in Figure 4.3b, late demonstrations can still guide the policy search toward the right solution. You can also see this effect in the reward curves shown in Figure 4.3c. While giving the via-points after some number of iterations can still cause a significant jump in reward, the size of this jump decreases for later iterations. After some time, the modified search distribution will even decrease the performance of the learning process. This effect can be attributed to a mismatch of the chosen prior distribution, i.e. the current policy, and the optimal prior distribution which is the unknown policy according to which the human teacher is sampling his via-points. As the learning process converges to a suboptimal policy, it is impossible to sample trajectories that adhere to both, the learned policy as well as the specified via-point. Depending on the value of the variance parameter, the learner can then either sample degenerate trajectories from low-probability areas of the current policy or ignore the given suggestion. To avoid this effect, via-points should always be given while the uncertainty in the current policy is relatively high in comparison to the deviation of the via-points from this policy. Note that in this experiment, the via-points are already known from the start even if the agent doesn't utilize them. This allowed us to analyze the effect of giving late demonstrations without having to account for the human factor. However, in practice, late demonstrations should be given depending on trajectories sampled from the current policy. This way, the user can actively shape the trajectory and guide the learning process to the right solution. We illustrate this in Figure 4.4 by showing how to change the policy with via-points in order to directly change the way the letter *w* is being drawn.

Evaluation of learning with linear models

For the third experiment, we investigated the properties of learning a parametric generalization of the above task with a linear via-point model. We are looking at learning a model that can recreate words with respect to rotation and scaling of the target image. Note

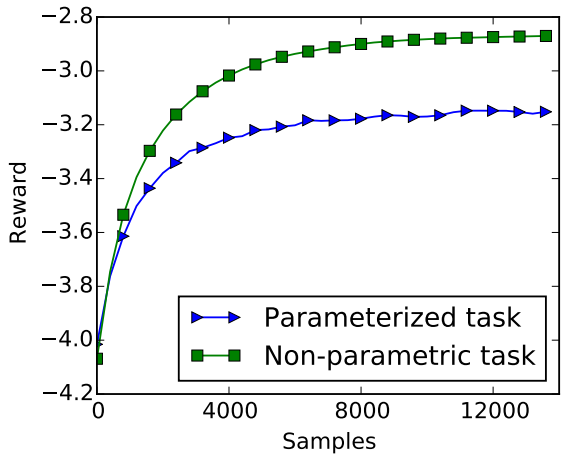


Figure 4.5: Reward obtained by contextual REPS shows that models of via-points can be used as an effective initialization. Rewards on the non-parametric task are displayed for comparison.

that similar kinds of parametric tasks can be found in practice, where the parameters often denote the location and orientation of an object. We are assuming that good features are known as this would be a necessity for obtaining a via-point model in practice. In this case we are using the feature vector $(s_x cos(\theta), s_y cos(\theta), s_x sin(\theta), s_y sin(\theta), 1)$ where s_x and s_y represent the scaling and lie between 0.6 and 1.4 and θ represents the rotation of the target image which lies between -45 and 45 degrees. The linear via-point models are then given w.r.t. these features and are used for learning a linear Gaussian policy using locally linear weighted regression and contextual REPS. Figure 4.5 shows that the initial policy adapts to the task parameters and that it can be used in conjunction with contextual REPS to obtain a similar reward curve for arbitrary rotations and scaling as for a single instance of the task.

4.5 Summary

We introduced an approach to utilize partial demonstrations to interactively guide the policy search process. We have shown that our approach of using soft via-points significantly outperforms continuous demonstrations when used to initialize the learning process and provides a more effective way to teach the robot to solve the tasks. Furthermore, our results show that our approach can be used to guide the learning process in an interactive

way, utilizing the knowledge gained from observing the robot to change specific aspects of the policy. Finally, we have shown the ability to generalize to different version of a particular task using linear models of via-points.

The work in this chapter represents one instance of an imitation learning approach which reasons about likely future observations in order to use a given demonstration more effectively and to allow the agent to learn from a very small amount of demonstration data. The approach uses a trajectory-based approach to reason about likely future observations and a reinforcement learning signal to disambiguate between trajectories that comply with the expert's demonstrations. If we wish to learn a more general policy that is able to adapt to unforeseen situations, we need to choose a different approach that does not rely on trajectory representations but rather on learned knowledge about the environment's dynamics. Furthermore, if we don't have a reward oracle that allows us to disambiguate between candidate policies, we furthermore need to develop an alternative measure of "goodness" based on the demonstration signal alone. In the next chapter, we will develop such an approach and lay the foundation for the remainder of this dissertation. In chapter 7, we will revisit a similar peg-in-hole experiment using via-point demonstrations, drawing further connections to the work presented in this chapter.

CHAPTER 5
A TEMPORAL DIFFERENCE APPROACH TO STATE AWARE IMITATION LEARNING

5.1 Motivation

We now consider a more general imitation learning set-up. We consider an agent that is learning a stationary policy, for example a policy represented by a neural network, to reproduce the behavior exhibited by the expert as accurately as possible, using only a very small amount of demonstration data. The central question to be addressed by any such imitation learning method is how the agent should act in situations that are unlike the ones that can found as part of the demonstration set.

In the previous chapter, this question could be answered using a reward signal what was given to us. Now, as we consider a setting where no such reward is available, the question is more ambiguous. Without an additional learning signal, we need to make assumptions about the expert's intent in order to generalize to unseen states. The assumption we will make here is that the states the expert visited when recording demonstrations are desirable while deviations from these states are undesirable. In more practical terms, this can also be motivated from the perspective of addressing the problem of accumulating errors, a known problem for the likely most straight-forward imitation learning approach, Behavioral Cloning [84]. In Behavioral Cloning, the agent uses a supervised loss to learn a mapping from states to actions directly, using demonstrated states and actions as training data. In practice, an agent trained with Behavioral Cloning often performs well, especially when the amount of training data is sufficiently large; however, when the demonstrations are relatively sparse, other approaches can significantly outperform it. Ross and Bagnell [91] formalize a likely reason for this: Behavioral Cloning violates a key assumption of

supervised learning as future inputs are affected by past predictions. The result is that mistakes lead the agent to states that differ from demonstrated states and thus future inputs on which the agent is likely to make further mistakes. The agent deviates further from the demonstration data and thus errors accumulate. To avoid deviations, we train the agent to explicitly attempt to match the expert's state-action joint distribution.

Joint-distributions matching requires the agent to reason about its future observations and, as environment dynamics are unknown, requires the agent to learn from samples it collects in a self-supervised way. Prior approaches achieve this in an indirect way, such as learning a reward function [73, 1], or by using an adversarial objective [39]. We refer to Chapter 3 for a survey of such methods. Here, we introduce State Aware Imitation Learning (SAIL, previously published in Schroecker and Isbell [96]), a more direct approach to joint-distribution matching based on the principle of maximum likelihood. The maximum-likelihood objective can be written as:

$$\text{argmax}_\theta \sum_{i=0}^{N} \log \pi_\theta(\overline{a}^{(i)}|\overline{s}^{(i)}) + \log d^{\pi_\theta}(\overline{s}^{(i)}). \tag{5.1}$$

Where we assume N demonstrated state-action pairs $\overline{s}^{(i)}, \overline{a}^{(i)}$. To optimize the objective, we will introduce an approximation of the full maximum-likelihood gradient

$$\sum_{i=0}^{N} \nabla_\theta \log \pi_\theta(\overline{a}^{(i)}|\overline{s}^{(i)}) + \nabla_\theta \log d^{\pi_\theta}(\overline{s}^{(i)}). \tag{5.2}$$

Note that this equation differs from the supervised approach in which the second term is dropped and thus only considered implicitly. Maximum-likelihood on d^{π_θ} leads to the agent actively trying to reproduce observations seen in the demonstrations. Intuitively, if an agent trained with SAIL finds itself in a state similar to a demonstrated state it will prefer actions that are similar to the demonstrated action, yet it will also prefer to remain near demonstrated states where the trained policy is more likely to be accurate. An agent trained with SAIL will thus learn how to recover if it deviates from the demonstrated trajectories. While

the motivation of avoiding accumulating errors gives the second term an auxiliary function and while maximum likelihood on it does not uniquely identify a policy, results in the following chapters will indicate that it is in fact the more powerful term. In practice, matching the stationary state distribution alone often allows the agent to learn an effective policy, allows it to do so from observation alone and allows it to do so even when demonstrations take the form of via-points.

Maximum-likelihood on the state-distribution itself is difficult. Not only is the likelihood function itself unknown, reasoning over changes to it requires the agent to learn the long-term effects of each of its actions. We base our approach on the work of Morimura et al. [69], who estimate a gradient of the distribution of states observed when following the current policy using a least squares temporal difference learning approach and use their results to derive an alternative formulation of the policy gradient. We discuss this approach in detail in Section 5.2.1 and extend the idea to an online temporal difference learning approach in Section 5.2.2. This adaptation gives us greater flexibility for our choice of function approximator and also provides a natural way to deal with an additional constraint to the optimization problem, which we will introduce below. In Section 5.2.4, we describe the full SAIL algorithm in detail and show that the estimated gradient can be used to derive a principled and novel imitation learning approach. We then evaluate our approach on a tabular domain in Section 5.3.1, comparing our results to a purely supervised approach to imitation learning as well as to a sample based inverse reinforcement learning method [13]. In Section 5.3.2, we show that SAIL can successfully be applied to learn a neural network policy in a continuous bipedal walker domain and achieves significant improvements over supervised imitation learning in this domain.

5.2 Approach

SAIL is a gradient ascent based algorithm to finding the true maximum-likelihood estimate of the joint state-action distribution. The primary challenge is to estimate the gradient of

the (stationary) state distribution induced by following the current policy. We write this distribution as $d^{\pi_\theta}(s)$ and, under ergodicity assumptions typically made in Markov Decision Processes (see Chapter 2), this distribution exists and is unique. In a reinforcement learning context, the gradient of the logarithmic stationary state distribution, $\nabla_\theta \log d^{\pi_\theta}(s)$, has first been derived by Morimura *et al.* [69] who relate it to a generalized value function on the time-reversed MDP; however, the same result also immediately follows from earlier analysis [100] which derives the gradient of stationary distributions of finite Markov chains. We will first state these finding in Section 5.2.1, as they will be central to our algorithm, derive an online temporal-difference approach to estimating the gradient in Section 5.2.2 and use it in order to state the full SAIL algorithm in Section 5.2.4.

5.2.1 A temporal difference approach to estimating $\nabla_\theta \log d^{\pi}(s)$

We first review the work by Morimura et al. [69] who first discovered a relationship between the gradient $\nabla_\theta \log d^{\pi_\theta}(s)$ and value functions as used in the field of reinforcement learning. Morimura et al. show that the gradient can be written recursively and decomposed into an infinite sum so that a corresponding temporal difference loss can be derived. Here, we briefly restate those results.

By definition of the stationary state distribution, its gradient in a state s' can be written in terms of prior states s and actions a.

$$\begin{aligned} \nabla_\theta d^{\pi_\theta}(s') &= \nabla_\theta \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)ds, a \\ \nabla_\theta \log d^{\pi_\theta}(s') &= \frac{1}{d^{\pi_\theta}(s')} \nabla_\theta \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)ds, a \end{aligned} \tag{5.3}$$

Applying the log ratio trick, we obtain

$$\begin{aligned} \nabla_\theta \log d^{\pi_\theta}(s') &= \int p(s,a|s') \left(\nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s)\right) ds, a \\ 0 &= \int p(s,a|s') \left(\nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s) - \nabla_\theta \log d^{\pi_\theta}(s')\right) ds, a \end{aligned} \tag{5.4}$$

This can be seen as an expected temporal difference error over the previous state and action where the temporal difference error is defined as

$$\delta(s, a, s') := \nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s) - \nabla_\theta \log d^{\pi_\theta}(s') \tag{5.5}$$

Compared to the temporal difference found in reinforcement learning, the roles of state s and next state s' are reversed and the reward is replaced by the vector-valued characteristic eligibility $\nabla_\theta \log \pi_\theta(a|s)$. This opens the door for temporal difference approaches to approximate the desired state-distribution gradient. Morimura et al. point out that in order to do so, another constraint has to be taken into account. In order for the gradient to be a valid gradient of a probability distribution, the following has to hold:

$$E[\nabla_\theta \log d^{\pi_\theta}(s)] = 0 \tag{5.6}$$

In the original work, Morimura et al. derive a least squares estimator for $\nabla_\theta \log d^{\pi_\theta}(s')$ based on minimizing the expected squared temporal difference error and a penalty to enforce the constraint. The algorithm is then applied it to policy gradient reinforcement learning. Instead, we formulate an online update rule to estimate the gradient, allowing for non-linear approximations of the gradient but retaining convergence in the linear case, and use the estimated gradient to derive a novel imitation learning algorithm.

5.2.2 Online temporal difference learning for $\nabla_\theta \log d^{\pi}(s)$

Now, with the relationship between the state-distribution gradient and a vector-valued value function on the time-reversed Markov chain in mind, we can derive an online temporal difference update rule for SAIL and learn an approximation of the desired gradient. We first consider the non-discounted case and show that it has convergence properties that are similar to the case of average reward temporal difference learning [114]. Online temporal difference learning algorithms are computationally more efficient than their least squares

batch counter parts and are essential when using high-dimensional non-linear function approximations to represent the gradient. We furthermore show that online methods give us a natural way to enforce the constraint $E_{s\sim d^{\pi_\theta}}[\nabla_\theta \log d^{\pi_\theta}(s)] = 0$.

Online average-reward temporal-difference learning only approximates the value function up to an unknown constant shift. In reinforcement learning, this constant shift does not pose a problem as it does not affect the optimal policy. In our case, however, we need to consider it explicitly as a constant shift of the gradient changes the resulting policy. We will find that addressing the constant shift is the same as enforcing the constraint $E[\nabla_\theta \log d^{\pi_\theta}(s)] = 0$ which Morimura et al. considered more explicitly. To separate out the constant shift, we define the unknown constant shift vector c and then define $f^*(s) := \nabla_\theta \log d^{\pi}(s) + c$. Temporal difference learning then enables us to learn a parametric approximation $f_\omega(s) \approx f^*(s)$. The temporal-difference update rule based on taking action a in state s and transitioning to state s' is given by

$$\omega_{k+1} = \omega_k + \alpha \nabla_\omega f_\omega(s') \left(f_\omega(s) + \nabla_\theta \log \pi(a|s) - f_\omega(s')\right). \tag{5.7}$$

Note that if f_ω converges to an approximation of f^* then due to $E[\nabla_\theta \log d^{\pi_\theta}(s)] = 0$, we have $\nabla_\theta \log d^{\pi}(\overline{s}) \approx f_\omega(\overline{s}) - E_{s\sim d^{\pi_\theta}}[f_\omega(s)]$. The expectation can be estimated as a sample mean based on past interactions, allowing us to recover the unbiased state-distribution gradient.

While convergence of temporal difference methods is not guaranteed in the general case, some guarantees can be made in the case of linear function approximation $f_\omega(s) := \omega^T \phi(s)$ [114]. We note that $E[\nabla_\theta \log \pi(a|s)] = 0$ and thus for each dimension of θ the update can be seen as a variation of average reward temporal difference learning where the reward is replaced by $\nabla_\theta \log \pi(a|s)$ and f_ω is bootstrapped based on the previous state as opposed to the next. While the role of current and next state in this update rule are reversed and this might suggest that updates should be done in reverse, the convergence results by

Algorithm 2 State Aware Imitation Learning

function SAIL(ω, α_θ, α_ω, $\overline{S}$, $\overline{A}$)
  $\theta \leftarrow$ SupervisedTraining($\overline{S}$, $\overline{A}$)
  for $k \leftarrow 0$..#Iterations **do**
    $S_E, A_E \leftarrow$ CollectUnsupervisedEpisode(π_θ))
    $\omega \leftarrow \omega + \alpha_\omega \frac{1}{|S_E|} \sum_{s,a,s' \in \text{transitions}(S_E,A_E)} \left(f_\omega(s) + \nabla_\theta \log \pi_\theta(a|s) - f_\omega(s')\right) \nabla_\omega f(s'))$
    $\mu \leftarrow \frac{1}{|S_E|} \sum_{s \in S_E} f_\omega(s)$
    $\theta \leftarrow \theta + \alpha_\theta \left(\frac{1}{|\overline{S}|} \sum_{\overline{s},\overline{a} \in \text{pairs}(\overline{S},\overline{A})} \left(\nabla_\theta \log \pi_\theta(\overline{a}|\overline{s}) + (f_\omega(\overline{s}) - \mu)\right) + \nabla_\theta \log p(\theta) \right)$
  return θ

Tsitsiklis and Roy [114] are dependent only on the limiting distribution of following the sample policy on the domain which remains unchanged regardless of the ordering of updates and the convergence results still hold. We formally expand on this intuitive notion in the next section, but it is not required for understanding the full SAIL algorithm. As is usual in temporal-difference learning, convergence cannot be guaranteed when the method is used with non-linear function approximators and all results in this case are purely empirical.

Introducing a discount factor So far, we related the update rule to average reward temporal difference learning as this was a natural consequence of the assumptions we were making. However, in practice we found that a formulation analogous to discounted reward temporal difference learning may work better. While this can be seen as a biased but lower variance approximation to the average reward problem [115], a perhaps more satisfying justification can be obtained by reexamining the simplifying assumption that the distribution of the previous state $p(s_{-1})$ is equal to the stationary state distribution $d^{\pi_\theta}(s_{-1})$. An alternative simplifying assumption is that $p(s_{-1})$ is a mixture of the starting state distribution d_0 and the stationary state distribution $p(s_{-1}) = (1-\gamma)d_0(s_{-1}) + \gamma d^\pi(s_{-1})$ for

$\gamma \in [0, 1]$. In this case, Equation 5.4 has to be altered and we have

$$0 = \int p(s, a|s')\delta(s, a, s')ds, a.$$

$$\delta(s, a, s') := \gamma \nabla_\theta \log d^{\pi_\theta}(s) + (1 - \gamma)\nabla_\theta \log d_0(s) + \nabla_\theta \log \pi_\theta(a|s) - \nabla_\theta \log d^{\pi_\theta}(s') \tag{5.8}$$

Note that $\nabla_\theta \log d_0(s) = 0$ and thus we recover the discounted update rule

$$\omega_{k+1} = \omega_k + \alpha \nabla_\omega f(s') \left(\gamma f(s) + \nabla_\theta \log \pi(a|s) - f(s')\right) \tag{5.9}$$

In Chapter 7, we show another justification of introducing a discount factor γ, relating the objective to the policy gradient theorem.

5.2.3 Convergence with linear function approximation

We now show convergence of SAIL when using linear function approximation to approximate $\nabla_\theta \log d^{\pi_\theta}(s)$. The proof is largely implied by the seminal findings of Tsitsiklis and Roy [114] who analyze convergence of average reward temporal difference learning. In this part we phrase the SAIL algorithm in their framework and show that the convergence results still apply. Notation in this section is partially inconsistent with the remainder of this dissertation to be more directly comparable to Tsitsiklis and Van Roy. Given a vector of rewards for each state g, Tsitsiklis and Van Roy define the differential value vector for $J^* := \sum_{t=0}^{\infty}(g - \mu^* e)$ where $\mu^* = E_i[g(i)]$ is the average reward under the stationary state distribution and e corresponds to the vector with all entries equal to 1. Learning a linear approximation $J_\omega = \omega\Phi$ with parameters ω and a matrix Φ with each row corresponding to the features $\phi(s)$ of each state s, they show that

1. The average cost temporal difference update of

$$\omega^{(k+1)} = \omega^{(k)} + \alpha\phi(s)\left(g(s) + \omega^{(k)}\phi(s_{+1}) - \omega^{(k)}\phi(s)\right)$$

where s_{+1} denotes the next state, converges to ω^* where ω^* is the unique solution to $E[g(s) + \omega^{(k)}\phi(s_{+1}) - \omega^{(k)}\phi(s)|q(s)] = 0$ and $q(s)$ denotes the limiting distribution of the samples used to perform this update. (Theorem 2)

2. Defining the backup operator $TJ = g - \mu^* e + PJ$ where P denotes the transition probability matrix of the MDP. If $q(s)$ is the stationary state distribution of the MDP, the unique solution corresponds to the projected fix point of the $\Phi\omega^* = \Pi T(\Phi\omega^*)$ where Π denotes the projection operator performing the shortest projection onto the space that can be represented by linear combinations of the chosen features.

3. Any fix point J of the backup operator T is equal to $J^* + ce$ for an unknown constant c.

We now show that SAIL performs average cost temporal difference learning for each dimension i to estimate $\nabla_{\theta_i} \log \pi_\theta(a|s)$. We define the row corresponding to state s in the cost vector g to be $E[\nabla_{\theta_i} \log \pi_\theta(a_{-1}|s_{-1})|p(s_{-1}, a_{-1}|s)]$ and first make the observation that this implies $\mu^* = 0$, as

$$\begin{aligned}
\mu_i^* &= \int d^{\pi_\theta}(s) p(a_{-1}, s_{-1}|s) \nabla_{\theta_i} \log \pi_\theta(a_{-1}|s_{-1}) ds, a_{-1}, s_{-1} \\
&= \int d^{\pi_\theta}(s_{-1}) \int \pi_\theta(a_{-1}|s_{-1}) \nabla_{\theta_i} \log \pi_\theta(a_{-1}|s_{-1}) da_{-1} ds_{-1} \\
&= \int d^{\pi_\theta}(s_{-1}) \cdot 0 ds_{-1} = 0.
\end{aligned} \tag{5.10}$$

Thus we observe that the SAIL update rule in the linear case

$$\omega_i^{(t+1)} = \omega_i^{(t)} + \alpha\phi(s)(\nabla_{\theta_i} \log \pi_\theta(a|s) + \omega_i \phi(s_{-1}) - \omega_i \phi(s)) \tag{5.11}$$

corresponds to average reward temporal difference learning on the Markov chain induced by the reverse transition probabilities $p(s_{-1}|s, \pi_\theta)$ and Theorem 1 of [114] holds. Next, we note that the samples are observed by following the forward transitions and the limiting distribution of samples collected this way is equal to the stationary state distribution of

Markov chain induced by the time-reversed transition dynamics as shown by Theorem 1 in Morimura *et al.* As per the above, the update thus converges to a projected fix point $J_i \approx J_i^* + ce$ for each dimension i. We now note that as per definition and using [69], we have

$$J^* = \sum_{t=0}^{\infty}(g - \mu^* e) = \sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(s_{-t}|a_{-t}) = \nabla_\theta \log d^{\pi_\theta}(s) \tag{5.12}$$

and thus

$$\nabla_\theta \log d^{\pi_\theta}(s) = J(s) - E[J(s)|d^\pi(s)]. \tag{5.13}$$

This shows that the SAIL update rule combined with the subsequent adjustment for the mean converges to an estimate of the $\nabla_\theta \log d^{\pi_\theta}(s)$ in the case of linear function approximation.

5.2.4 State aware imitation learning

Based on this estimate of $\nabla_\theta \log d^{\pi_\theta}$, we can now derive the full State Aware Imitation Learning algorithm. SAIL matches the expert's joint distribution by following the full maximum likelihood gradient as defined in Equation 5.1. The gradient decomposes into two parts (or three with prior $\nabla_\theta \log p(\theta)$):

$$\nabla_\theta \log \rho^{\pi_\theta}(\overline{s}, \overline{a}) = \nabla_\theta \log \pi_\theta(\overline{a}|\overline{s}) + \nabla_\theta \log d^{\pi_\theta}(\overline{s}) \tag{5.14}$$

The first makes up the gradient used for gradient descent based supervised learning and can usually be computed analytically while the second term can be estimated via temporal-difference learning as described in the previous section. The full SAIL algorithm thus maintains a current policy as well an estimate of $\nabla_\theta \log d^{\pi_\theta}(\overline{s}|\theta)$ and iteratively

1. Collects M unsupervised state and action samples $s^{(i)}$ and $a^{(i)}$ from the current policy,

2. Updates the gradient estimate using the collected samples with Equation 5.7

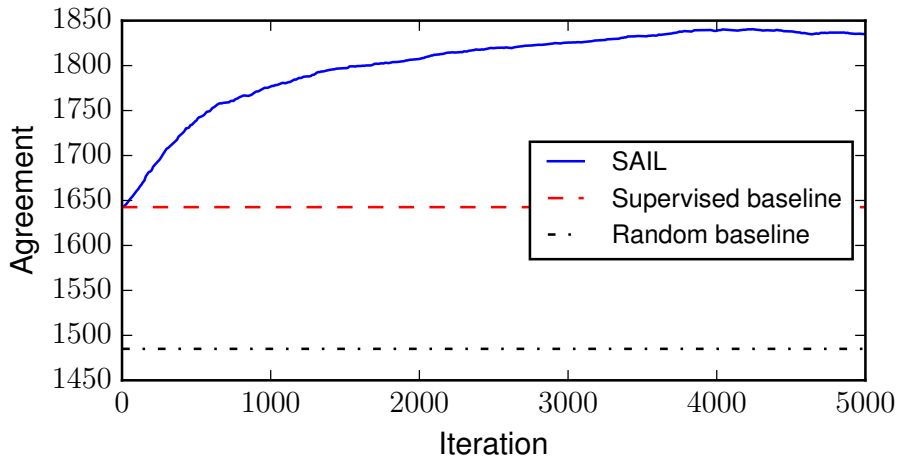


Figure 5.1: The sum of probabilities of taking the optimal action double over the baseline. While the optimal policy with regards to the state-action-distribution matching objective does not need to be identical to the optimal policy with regards to the discounted reward objective, the policy learned by SAIL takes the optimal action more often than a policy trained via Behavioral Cloning

3. Estimates $E[f_\omega(s)]$ using the sample mean of the unsupervised states or an exponentially moving sample mean

$$\mu := \frac{1}{M} \sum_{i=0}^{M} f_\omega(s^{(i)})$$

4. Updates the current policy using the estimated gradient $f_\omega(s) - \mu$ as well as the analytical gradients for $\nabla_\theta \log p(\theta)$ and $\nabla_\theta \log \pi_\theta(\overline{a}^{(i)}|\overline{s}^{(i)})$. The SAIL gradient is given by

$$\sum_{i=0}^{N} \left(f_\omega(\overline{s}^{(i)}) - \mu + \nabla_\theta \log \pi_\theta(\overline{a}^{(i)}|\overline{s}^{(i)}) \right)$$

The full algorithm is stated in Algorithm 2.

5.3 Evaluation

We evaluate our approach on two domains. The first domain is a harder variation of the tabular racetrack domain first used in Boularias *et al.* [13] with 7425 states and 5 actions.

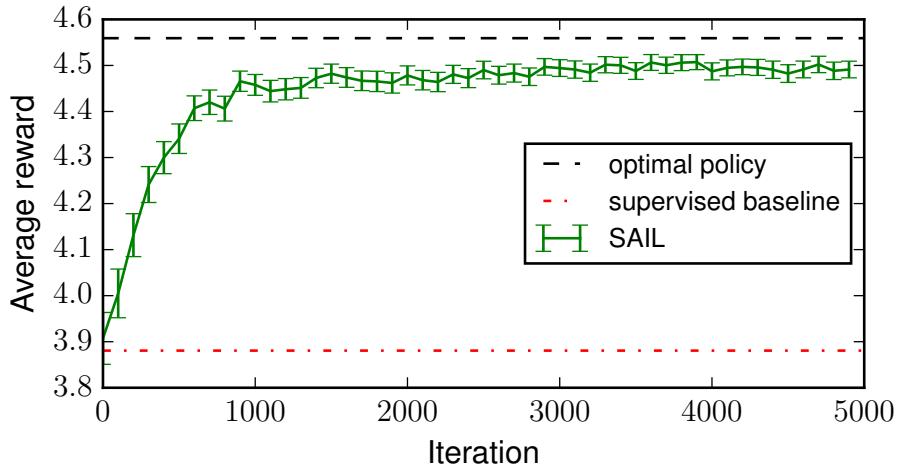


Figure 5.2: The average reward $(+/-2\sigma)$ obtained after 5000 iterations. The reward obtained by an agent trained with SAIL is much closer to the optimal policy.

We first use this domain to show that SAIL can improve on the policy learned by a supervised baseline and learn to act in states the policy representation does not generalize to. Subsequently, we evaluate sample efficiency of an off-policy variant of SAIL. The tabular representation allows us to compare the results to RelEntIRL [13] as a baseline without restrictions arising from the chosen representation of the reward function. The second domain we use is a noisy variation of the bipedal walker domain found in OpenAI gym [15]. We use this domain to evaluate the performance of SAIL on tasks with continuous state and action spaces using neural networks to represent the policy as well as the gradient estimate and compare it against the supervised baseline using the same representations.

5.3.1 Racetrack domain

We first evaluate SAIL on a tabular domain. While tabular domains are in many ways easier to solve than domains that require the use of function approximation, they are uniquely challenging for behavioral cloning. The tabular nature prohibits the agent to generalize based on representation alone and thus ensures that any generalization to other states is the consequence of the agent learning about the environment dynamics. A supervised method cannot learn a good policy in unseen states. The racetrack domain is a more difficult

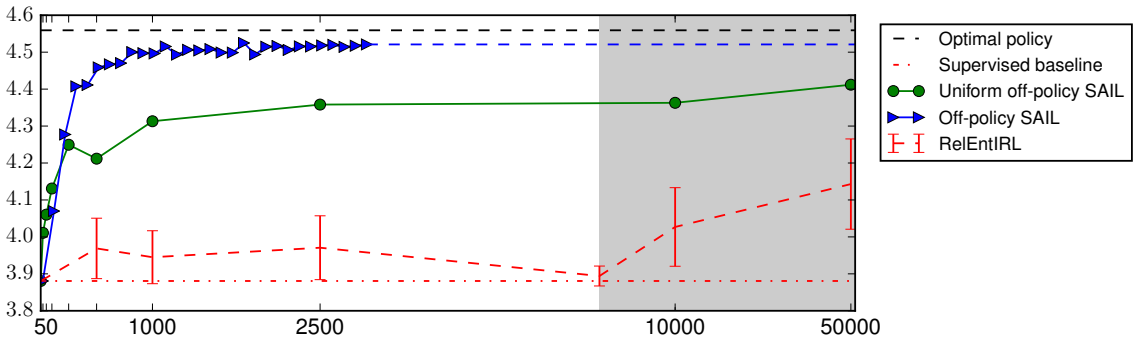
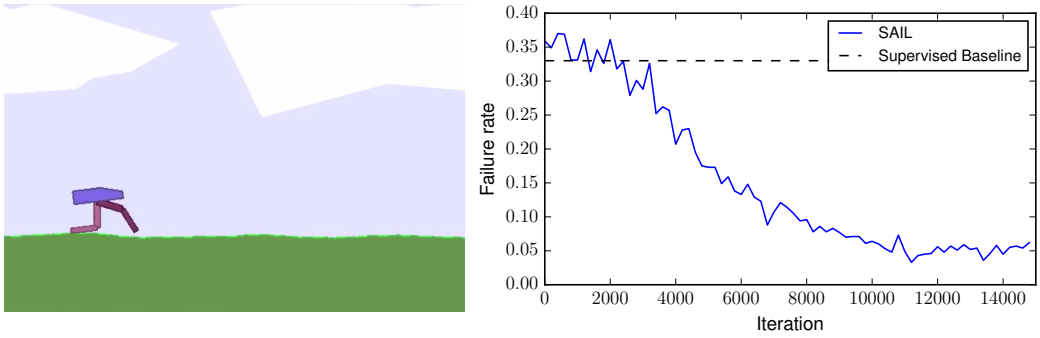


Figure 5.3: Reward obtained using off-policy training. SAIL learns a near-optimal policy using only 1000 sample episodes. The scale is logarithmic on the x-axis after 5000 iterations (gray area).

variation of the tabular domain used by Boularias *et al.* [13] and consists of a grid with 33 by 9 possible positions. Each position has 25 states associated with it, encoding the velocity (-2, -1, 0, +1, +2) in the x and y direction which dictates the movement of the agent at each time step. The domain has 5 possible actions allowing the agent to increase or reduce its velocity in either direction or to keep its current velocity. Randomness is introduced to the domain using the notion of a failure probability which is set to be 0.8 if the absolute velocity in either direction is 2 and 0.1 otherwise. The goal of the agent is to complete a lap around the track without going off-track which we define to be the area surrounding the track ($x = 0, y = 0, x > 31$ or $y > 6$) as well as the inner rectangle ($2 < x < 31$ and $2 < y < 6$). Note that unlike in [13], the agent has the ability to go off-track as opposed to being constrained by a wall and has to learn to move back on track if random chance makes it stray from it. Furthermore, the probability of going off-track is higher as the track is more narrow in this variation of the domain. This makes the domain more challenging to learn using imitation learning alone.

For all our experiments, we use a set of 100 episodes collected from an oracle. To measure performance, we assign a score of -0.1 to being off-track, a score of 5 for completing the lap and -5 for crossing the finish line the wrong way. Note that this score is not used during training but is purely used to measure performance in this evaluation. We also use



(a) (b)

Figure 5.4: a) The bipedal walker has to traverse the plain, controlling the 4 noisy joint motors in its legs. b) Failure rate of SAIL over 1000 traversals compared to the supervised baseline measured. After 15000 iterations, SAIL traverses the plain far more reliably than the baseline.

this score as a reward to derive an oracle.

On-policy results

For our first experiment, we compare SAIL against a supervised baseline. As the oracle is deterministic and the domain is tabular, this means taking the optimal action in states encountered as part of one of the demonstrated episodes and uniformly random actions otherwise. For the evaluation of SAIL, we initialize the policy to the supervised baseline and use the algorithm to improve the policy over 5000 iterations. At each iteration, 20 unsupervised sample episodes are collected to estimate the SAIL gradient, using plain stochastic gradient descent with a learning rate of 0.1 for the temporal difference update and RMSprop with a a learning rate of 0.01 for updating the policy. Figure 5.2 shows that SAIL stably converges to a policy that significantly outperforms the supervised baseline. While we do not expect SAIL to act optimally in previously unseen states but to instead exhibit recovery behavior, it is interesting to measure on how many states the learned policy agrees with the optimal policy using a soft count for each state based on the probability of the optimal action. Figure 5.1 shows that the amount of states in which the agent takes the optimal action roughly doubles its advantage over random chance and that the learned

behavior is significantly closer to the optimal policy on states seen during execution.

Off-policy sample efficiency

For our second experiment, we evaluate the sample efficiency of SAIL by reusing previous sample episodes. As a temporal difference method, SAIL can be adapted using any off-policy temporal difference learning technique. In this work we elected to use truncated importance weights [71]. We evaluate the performance of SAIL collecting one new unsupervised sample episode in each iteration, reusing the samples collected in the past 19 episodes and compare the results against our implementation of Relative Entropy IRL [13]. We found that the importance sampling approach used by RelEntIRL makes interactions obtained by a pre-trained policy ineffective when using a tabular policy[1] and thus collect samples by taking actions uniformly at random. For comparability, we also evaluated SAIL using a fixed set of samples obtained by following a uniform policy. In this case, we found that the temporal-difference learning can become unstable in later iterations and thus decay the learning rate by a factor of 0.995 after each iteration.

We vary the number of unsupervised sample episodes and show the score achieved by the trained policy in Figure 5.3. The score for RelEntIRL is measured by computing the optimal policy given the learned reward function. Note that this requires a model that is not normally available. We found that in this domain depending on the obtained samples, RelEntIRL has a tendency to learn shortcuts through the off-track area. Since small changes in the reward function can lead to large changes in the final policy, we average the results for RelEntIRL over 20 trials and bound the total score from below by the score achieved using the supervised baseline. We can see that SAIL is able to learn a near optimal policy using a low number of sample episodes. We can furthermore see that SAIL using uniform samples is able to learn a good policy and outperform the RelEntIRL baseline reliably.

[1]The original work by Boularias et al. shows that a pre-trained sample policy can be used effectively if a trajectory based representation is used

5.3.2 Noisy bipedal walker

For our second experiment, we evaluate the performance of SAIL on a noisy variant of a two-dimensional Bipedal walker domain (see Figure 5.4a). The goal of this domain is to learn a policy that enables the simulated robot to traverse a plain without falling. The state space in this domain consists of 4 dimensions for velocity in x and y directions, angle of the hull, angular velocity, 8 dimensions for the position and velocity of the 4 joints in the legs, 2 dimensions that denote whether the leg has contact with the ground and 10 dimensions corresponding to lidar readings, telling the robot about its surroundings. The action space is 4 dimensional and consists of the torque that is to be applied to each of the 4 joints. To make the domain more challenging, we also apply additional noise to each of the torques. The noise is sampled from a normal distribution with standard deviation of 0.1 and is kept constant for five consecutive frames at a time. The noise thus has the ability to destabilize the walker. Our goal in this experiment is to learn a continuous policy from demonstrations, mapping the state to torques and enabling the robot to traverse the plain reliably. As a demonstration, we provide a single successful crossing of the plain. The demonstration has been collected from an oracle which has been trained on the bipedal walker domain without additional noise and is therefore not optimal and prone to failure. Our main metric for success on this domain is failure rate, i.e. the fraction of times that the robot is not able to traverse the plain due to falling to the ground. While the reward metric used in [15] is more comprehensive as it measures speed and control cost, it cannot be expected that a pure imitation learning approach can minimize control cost when trained with an imperfect demonstration that does not achieve this goal itself. Failure rate, on the other hand can always be minimized by aiming to reproduce a demonstration of a successful traversal as well as possible.

To represent our policy, we use a single shallow neural network with one hidden layer consisting of 100 nodes with tanh activation. We train this policy using a pure supervised approach as a baseline as well as with SAIL and contrast the results. During evaluation

and supervised training, the output of the neural network is taken to be the exact torques whereas SAIL requires a probabilistic policy. Therefore we add additional Gaussian noise, kept constant for 8 consecutive frames at a time.

To train the network in a purely supervised approach, we use RMSProp over 3000 epochs with a batch size of 128 frames and a learning rate of 10^{-5}. After the training process has converged, we found that the neural network trained with pure supervised learning fails 1650 times out of 5000 runs.

To train the policy with SAIL, we first initialize it with the aforementioned supervised approach. The training is then followed up with training using the combined gradient estimated by SAIL until the failure rate stops decreasing. To represent the gradient of the logarithmic stationary distribution, we use a fully connected neural network with two hidden layers of 80 nodes each using ReLU activations. Each episode is split into mini-batches of 16 frames. The $\nabla_\theta \log d^{\pi_\theta}$-network is trained using RMSprop with a learning rate of 10^{-4} whereas the policy network is trained using RMSprop and a learning rate of 10^{-6}, starting after the first 1000 episodes. As can be seen in Figure 5.4b, SAIL increases the success rate of 0.67 achieved by the baseline to 0.938 within 15000 iterations.

5.4 Summary

We introduced State Aware Imitation Learning (SAIL), a novel approach to imitation learning via state-action distribution matching and the first to optimize this objective by directly estimating the gradient of the maximum-likelihood objective. Imitation learning has long been a topic of active research; however, naive supervised learning has a tendency to lead the agent to states in which it cannot act with certainty. State-action-distribution matching often fares better by teaching the agent to stick to states that it has observed in the provided demonstrations. We show that SAIL is able to teach the agent to recover when deviating from such states and as a result teach the agent a more robust policy. We show that this enables the agent to learn from much sparser data. In this chapter, we explored

this on a bipedal walker domain, training the walker with a single demonstration. In the following chapters, we will build on this result and push sample-efficiency further, showing that state-action-distribution matching is sufficient to train the agent using even partial demonstrations such as demonstrated via-points.

The temporal-difference approach to estimating the state-distribution gradient has the advantage that it is fully model-free; however, it also has clear short-comings that arise as we set our sights on more complex domains. While the 2d bipedal walker can be solved using a neural network policy with a single hidden layer, more complex policy representations are needed on more difficult domains, often requiring tens of thousands or even millions of parameters. Learning a representation of the gradient requires us to learn an accurate representation for the partial derivative of each parameter, making the difficulty of the learning process scale linearly with the size of the policy. Not only is this a computational challenge, the additional variance also significantly increases the amount of required interactions with the environment. In the following chapters, we will thus build on the findings of this chapter and introduce two more scalable imitation learning approaches to estimate the same state-distribution gradient. We will show that the same principles can be used to derive imitation learning algorithms that are capable of learning from significantly fewer environment interactions and handle more complex environment dynamics.

CHAPTER 6

RECENT ADVANCES IN GENERATIVE MODELING

State aware imitation learning allows us to pursue a maximum-likelihood approach to state-distribution matching, leading to an effective approach to imitation learning which teaches the agent robust policies. Building on this approach, we will now focus on introducing methods that use these same principles while being more scalable and sample-efficient. To this end, we will introduce the concept of discounted long-term generative models. In the following chapters we will introduce two imitation learning algorithms, GPRIL (Chapter 7) and VDI (Chapter 9). GPRIL allows us to compute the same state-distribution gradient using a discounted long-term predecessor model to avoid representation of the gradient and training via temporal-difference learning. In this approach, the agent models likely past state-action pairs conditioned on an observed future state. The desired distribution can be defined as:

$$\mathcal{B}^{\pi}_{\gamma}(s, a|\overline{s}) := (1-\gamma)\sum_{j=0}^{\infty}\gamma^{j} q(s_t = s, a_t = a|s_{s+j} = \overline{s}) \tag{6.1}$$

We state this definition here only to motivate the need for advanced generative modeling techniques and will show the justification for this formulation in the following chapter. VDI combines long-term models with temporal-difference learning to achieve greater precision and stability, allowing for imitation learning with complex dynamics. The long-term model used in VDI is the time-reversal of $\mathcal{B}^{\pi}_{\gamma}$, modeling likely future states based on an observed state-action pair:

$$\mathcal{F}^{\pi}_{\gamma}(\overline{s}|s, a) := (1-\gamma)\sum_{j=0}^{\infty}\gamma^{j} p(s_{s+j} = \overline{s}|s_t = s, a_t = a) \tag{6.2}$$

Using long-term models allows us to bypass the problem of accumulating errors that is likely to arise with long-term predictions using single-step models. Past approaches to sequence prediction (e.g. [32, 54]) have shown such "jumpy" models to be both, more accurate and computationally faster. Note, however, that we will use simpler approaches than prior work on jumpy predictions as the target distribution changes while the agent is learning and thus learning speed and stability are at least as important as accurate predictions. Modeling long-term behavior and dynamics generally requires us to use a probabilistic representation. For time-reversed models, this is the case even if both, the policy and the environment dynamics, are deterministic. The tools for modeling complex target distributions such as $\mathcal{B}^{\pi}_{\gamma}$ and $\mathcal{F}^{\pi}_{\gamma}$ can be found among recent advances in generative modeling. The purpose of this chapter is to review such methods and lay the foundation for the following chapters.

A variety of approaches such as Generative Adversarial Networks [30], Variational Auto Encoders [46], autoregressive networks (e.g. Germain *et al.* [28], van den Oord *et al.* [120], and Van Den Oord *et al.* [119] and normalizing flows [21, 76] have been proposed which enable us to learn complex distributions and efficiently generate samples. Broadly, the field of generative models can be split into two categories: Implicit generative models are able to produce samples that are approximately distributed by the target-distribution, but do not learn an explicit representation of the density function. Density estimation approaches instead primarily focus on learning this function. Of the two approaches, implicit models have been paid significantly more attention in recent years.

6.1 Implicit Generative Models

Compared to successful, deep density estimators, trained implicit models tend to be computationally simpler and, when applied to images, tend to produce more visually appealing samples (e.g. [14]). Two main approaches have emerged in this class, Variational Autoencoders (VAEs) [46] and Generative Adversarial Networks (GANs) [30]. Both approaches,

GANs and VAEs, learn a generator function, a function which transforms latent variables sampled from a known distribution, typically a normal distribution, to values distributed by the target distribution. VAEs achieve this by training a probabilistic encoder (posterior), mapping training samples to latent variables. The generator serves as a decoder, transforming latent variables back to a reconstruction of training variables. A reconstruction loss which enforces meaningful reproduction is combined with a divergence-loss, shaping the distribution of latent variables and allowing latent variables to be sampled. Justifying the approach theoretically, the training procedure can be shown to maximize the evidence lower bound, a lower bound on the maximum-likelihood objective; although alternative justifications exist [38]. GANs, in contrast, are generally considered to produce the visually most impressive samples (e.g. [87, 14]) by using an adversarial training procedure. A discriminator is trained to identify samples generated by the generator among training samples. In parallel, the generator is updated to "fool" the discriminator, i.e. to maximize the discriminator's loss in an adversarial fashion.

Both methods have successfully been applied to predicting sequences step-by-step [9, 58] and VAEs in particular have been used for long-term prediction of future observations in benchmark reinforcement learning domains [32], a formulation similar to ours. Both training procedures, however, are also known to be prone to collapse [61, 90] and may not accurately model the target distribution itself [7]. In our work these issues are exacerbated as the target distribution changes over time as the agent updates its policy. In contrast, explicit density estimation methods allow for analytical computation of the maximum-likelihood gradient and tend to be more stable as a result. For this reason, we use explicit methods in both, Chapters 8 and 9, where a density model is strictly necessary as well as in the next chapter, where an implicit method could be used instead.

6.2 Autoregressive Models

Autoregressive models (e.g. [118, 28, 120, 119]) are capable of representing complex distributions $p(x); x \in \mathbb{R}^n$ by factoring the target distribution and learning a model for each conditional distribution.

$$p(x) = p_1(x_1) \prod_{i=1}^{N-1} p_{i+1}(x_{i+1}|x_1, \ldots, x_i) \tag{6.3}$$

While each p_i may be a simple distribution, for example a Gaussian or categorical distribution over comparatively few discrete values, the resulting joint distribution can be highly complex. Learning a separate model for each p_i, however, would be impractical if the number of features is even moderately large.

Masked autoencoders [28] provide a straight-forward approach to parameter sharing and allow representing each distribution to be represented by a single network. The approach models the distribution with a model similar to an auto-encoder, masking connections that violate the autoregressive property and reconstructing each x_i probabilistically based on $x_{0:i-1}$. Similar architectures have been proposed for more structured data. Images can be modeled either with a similarly masked convolutional network or using a recurrent neural network to predict each pixel based on previous pixels [120]. Wavenet [119] models audio-data with temporal 1d convolutions, predicting a categorical distribution over audio-samples conditioned on previous time-steps.

Using the reparameterization trick, autoregressive models can be seen as a transformation of latent variables, similar to the generator in VAEs or GANs; however, unlike in implicit models, this transformation is bijective. A Gaussian autoregressive model, i.e. an autoregressive model which predicts mean and variance of a Gaussian distribution to model each factorial p_i, can be seen as a learned affine transformation of a latent variable

$z \sim p_z(z)$, where p_z is Gaussian. The transformation can be written as

$$x_i = \mu_i(x_{0:i-1}) + \sigma_i(x_{0:i-1})z. \tag{6.4}$$

Here, μ and σ are learned functions which obey the autoregressive property. While autoregressive models can model complex distributions, there are many situations in which their expressivity is limited. For example, when using Gaussian conditional distributions, the first feature x_0 will invariably be distributed by a Gaussian. In our work, we thus use autoregressive flows. Autoregressive flows chain multiple such transformations to achieve greater expressivity at the cost of computational and representational complexity.

6.3 Normalizing Flows

Central to normalizing flows (e.g. [21, 76, 47, 55]) is the change of variable technique which allows for computing the density $p_x(x)$ of a random variable x, if x is the result of a an invertible transformation $f : \mathbb{R} \to \mathbb{R}$ as

$$\log p_x(x) = \log p_z(f^{-1}(x)) + \log \det(|J(f^{-1}(x))|) \tag{6.5}$$

Efficient computation of the density requires us to be able to efficiently compute the inverse $f^{-1}(z)$ as well as the determinant of its Jacobian. The most common choice is to restrict the transformation f to affine transformations such as described in the previous section (although recent work shows that more complex transformations are possible and can be helpful [40]). If μ and σ are autoregressive, the inverse can be recovered efficiently as $z_i = \frac{x_i}{\sigma_i(x_{0:i-1})} - \mu_i(x_{0:i-1})$ is parallelizable if x is known. Furthermore, due to the autoregressive nature of μ and σ, the Jacobian of f^{-1} is triangular which allows for efficient computation of the determinant $\log \det(|J(f^{-1}(x))|) = -\sum_{i=-}^{N} \log \sigma_i(x_{0:i-1})$.

Being able to compute the likelihood $p_x(x)$ in closed form and, as a direct result, its gradient $\nabla \log p_x(x)$, Normalizing Flows can be trained via maximum-likelihood. Apply-

ing a single autoregressive affine transformation to a normal distribution leads to a model equivalent to an autoregressive model with Gaussian conditional distributions. However, the formulation allows for multiple such transformations to be chained. This model, called Masked Autoregressive Flow (MAF), has been proposed by Papamakarios *et al.* [76] and evaluated on density estimation benchmark tasks and simple images. We will use MAFs to learn discounted long-term predecessor models in the next chapter.

RealNVP [21] and its successors [47, 55] similarly consist of a series of affine transformations but utilizes a block-autoregressive structure for μ and σ. Each affine transformation transforms half the features dependent on the other half of the features. RealNVP defines such transformations in a way that is particularly well-suited for images and evaluate the approach on realistic image datasets. In Chapters 8 and 9 we will use a simplified version of RealNVPs which utilize the same block-autoregressive structure, applied to non-image features. Here, we simply alternate predicting an affine transformation of even-numbered features based on odd-numbered features and predicting an affine transformation of odd-numbered features based on even-numbered features. The original model furthermore introduces a batch-norm transformation which we find is not necessary for non-image data. The choice of this model over MAFs is one of convenience: while block-autoregressive transformations are less expressive than the fully autoregressive transformations used in MAFs, their gradients are less likely to explode and it is easier to find hyper-parameters such that the training process remains stable.

CHAPTER 7

GENERATIVE PREDECESSOR MODELS FOR IMITATION LEARNING

7.1 Motivation

In Chapter 5, we introduced a novel imitation learning approach that is able to reason explicitly about how to change the policy in order to reproduce the distribution of states observed in a given set of expert demonstrations. This allows the agent to learn robust behavior and learn effective policies from only a small set of demonstrations. However, the presented approach has limitations in terms of scaling with respect to the number of parameters of the policy as well as concerning sample-efficiency regarding the number of required interactions of the agent with the environment. This limits the real-world applicability of the presented algorithm. In this chapter, we aim to build on SAIL and introduce a novel algorithm which similarly follows a maximum-likelihood approach considering the joint distribution over states and actions while resolving the issues of scaling and sample-efficiency. To this end, we will make use of recent advances in generative modeling. The resulting algorithm is GPRIL (Generative Predecessor Models for Imitation Learning), previously published in Schroecker *et al.* [97]. Before deriving this approach based on the methodology introduced in deriving SAIL, we will first motivate our algorithm in an intuitive way, using the following simple core insight: Augmenting the training set with state-action pairs that, under the current behavior, are likely to eventually lead the agent to states demonstrated by the expert is an effective way to train corrective behavior and to prevent accumulating errors.

Recent advances in generative modeling, such as Goodfellow *et al.* [30], Kingma and Welling [46], van den Oord *et al.* [120], Van Den Oord *et al.* [119], and Dinh *et al.* [21], have shown great promise at modeling complex distributions and can be used to reason

probabilistically about such state-action pairs. Specifically, we propose to utilize Masked Autoregressive Flows [76] to model long-term predecessor distributions, i.e. distributions over state-action pairs which are conditioned on a state that the agent will see in the future. Predecessor models have a long history in reinforcement learning (e.g. Peng and Williams [79]) with recent approaches using deep networks to generate off-policy transitions [23, 75] or to reinforce behavior leading to high-value states [31]. Here, we use predecessor models to derive a principled approach to state-distribution matching and propose the following imitation learning loop:

1. Interact with the environment and observe state, action as well as a target state. To encode long-term corrective behavior, these states should be multiple steps apart.

2. Train a conditional generative model to produce samples like the observed state-action pair when conditioned on the observed target state.

3. Train the agent in a supervised way, augmenting the training set using data drawn from the model conditioned on the demonstrations. The additional training data shows the agent how to reach demonstrated states, enabling it to recover after deviating from expert behavior.

This sketches out an algorithm that intuitively learns to reason about the states it will observe in the future. In Section 7.2, we derive this algorithm from first principles as a maximum likelihood approach to matching the state-action distribution of the agent to the expert's distribution. In Section 7.3, we compare our approach to a state-of-the-art imitation learning method [39] and show that it matches or outperforms this baseline on our domains while being significantly more sample efficient. Furthermore, we show that GPRIL can learn using demonstrated states alone, allowing for a wider variety of methods to be used to record demonstrations. Together these properties are sufficient to allow GPRIL to be applied in real-world settings, which we demonstrate in Section 7.3.3. To our knowledge this is the first instance of dynamic, contact-rich and adaptive behavior being taught solely

Algorithm 3 Generative Predecessor Models for Imitation Learning (GPRIL)

function GPRIL($N_{\mathcal{B}}$, N_{π})
  for $i \leftarrow 0..$#Iterations **do**
    for $k \leftarrow 0..N_{\mathcal{B}}$ **do**
      for $n \leftarrow 0..$#BatchSize **do**
        Sample $s_t^{(n)}, a_t^{(n)}$ from replay buffer
        Sample $j \sim Geom(1-\gamma)$
        Sample $s_{t+j}^{(n)}$ from replay buffer
      Update ω_s using gradient $\sum_{n=0}^{N_{\mathcal{B}}} \nabla_{\omega_s} \log \mathcal{B}^s_{\omega_s}(s_t^{(n)}|s_{t+j}^{(n)})$
      Update ω_a using gradient $\sum_{n=0}^{N_{\mathcal{B}}} \nabla_{\omega_a} \log \mathcal{B}^a_{\omega_a}(a_t^{(n)}|s_t^{(n)}, s_{t+j}^{(n)})$
    for $k \leftarrow 0..N_{\pi}$ **do**
      for $n \leftarrow 0..$#BatchSize **do**
        Sample $\overline{s}^{(n)}, \overline{a}^{(n)}$ from expert demonstrations
        Sample $s^{(n)} \sim \mathcal{B}^s_{\omega_s}(\cdot|\overline{s}^{(n)})$, $a^{(n)} \sim \mathcal{B}^a_{\omega_a}(\cdot|s^{(n)}, \overline{s}^{(n)})$
      Update θ using gradient $\sum_{n=0}^{N_{\pi}} \beta_{\pi} \nabla_{\theta} \log \pi_{\theta}(\overline{a}^{(n)}|\overline{s}^{(n)}) + \beta_d \nabla_{\theta} \log \pi_{\theta}(a^{(n)}|s^{(n)})$

using the kinesthetic-teaching interface of a collaborative robot, without resorting to tele-operation, auxiliary reward signals, or manual task-decomposition.

7.2 Approach

The algorithm outlined above, provides us with an intuitive framework for using discounted long-term predecessor models to augment our training set and achieve robust imitation learning from a very small number of demonstrations. To derive this algorithm from first principles, we again turn to the state-action distribution matching via maximum-likelihood. We first derive the gradient of the logarithmic stationary state distribution based on samples from the discounted long-term predecessor distribution, paving the way for approximation of this gradient without representing it by a neural network as in Chapter 5. Next, we propose to learn a model of this distribution from self-supervised roll-outs in the environment. This will then allow us to describe the full GPRIL algorithm. Finally, we give an alternative justification for the approach based on the policy gradient theorem

7.2.1 Approximating the state-distribution gradient

We will show that the samples drawn from a long-term predecessor distribution conditioned on $\overline{s}$ enable us to estimate the gradient of the logarithmic state distribution $\nabla_\theta \log d^{\pi_\theta}(\overline{s})$ and, later, to match the agent's state-action-distribution to that of the expert. To achieve this goal, we can, similar to the derivation of SAIL, utilize the fact that the stationary state distribution of a policy can be defined recursively in terms of the state distribution at the previous time step. This recursion allows us to derive the gradient of the stationary state distribution as follows (we refer to Section 5.2.1 and Morimura *et al.* [69] for the full derivation of this gradient):

$$d^{\pi_\theta}(\overline{s}) = \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s_{t+1} = \overline{s}|s_t = s, a_t = a)ds, a \tag{7.1}$$

$$\nabla_\theta \log d^{\pi_\theta}(\overline{s}) = \int q^{\pi_\theta}(s_t = s, a_t = a|s_{t+1} = \overline{s}) \left(\nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s)\right) ds, a \tag{7.2}$$

In deriving SAIL, we used this property to arrive at a temporal difference approach to estimating the gradient. Here, we instead use the recursive nature of this gradient to unroll the gradient indefinitely. This process is cumbersome and we will state the result first before deriving it in Section 7.2.5:

$$\nabla_\theta \log d^{\pi_\theta}(\overline{s}) = \lim_{T\to\infty} \int \sum_{j=0}^{T} q^{\pi_\theta}(s_t = s, a_t = a|s_{t+j+1} = \overline{s}) \nabla_\theta \log \pi_\theta(a|s) ds, a \tag{7.3}$$

The derivation of our approach now rests on two key insights: First, in ergodic Markov chains, such as the ones considered in our setting, decisions that are made at time t affect the probability of seeing state $\overline{s}$ at time $t + j$ more strongly if j is small. In the limit, as $j \to \infty$, the expectation of the gradient $\nabla_\theta \log \pi_\theta(a_t|s_t)$ vanishes and the decision at time t only adds variance to the gradient estimate. Introducing a discount factor γ similar to common practice in reinforcement learning [107] places more emphasis on decisions that

are closer in time and can thus greatly reduce variance. Second, by introducing a discount factor, the effective time-horizon is now finite. This allows us to replace the sum over all states and actions in each trajectory with a scaled expectation over state-action pairs. Here, a higher discount factor implies a larger time-horizon and a closer approximation of the state-distribution matching objective while a lower discount factor allows for easier training. Formally, we can write this as follows and arrive at our main result:

$$\begin{aligned}\nabla_\theta \log d^{\pi_\theta}(\overline{s}) &\approx \int \sum_{j=0}^{\infty} \gamma^j q^{\pi_\theta}(s_t = s, a_t = a | s_{t+j+1} = \overline{s}) \nabla_\theta \log \pi_\theta(a|s) ds, a \\ &\propto \mathbb{E}_{s,a \sim \mathcal{B}^{\pi_\theta}(\cdot,\cdot|\overline{s})} \left[\nabla_\theta \log \pi_\theta(a|s) \right]\end{aligned} \tag{7.4}$$

where $\mathcal{B}^{\pi_\theta}$ corresponds to the long-term predecessor distribution modeling the distribution of states and actions that, under the current policy π_θ, will eventually lead to the given target state $\overline{s}$:

$$\mathcal{B}^{\pi_\theta}(s, a|\overline{s}) := (1-\gamma) \sum_{j=0}^{\infty} \gamma^j q^{\pi_\theta}(s_t = s, a_t = a | s_{t+j+1} = \overline{s}) \tag{7.5}$$

7.2.2 Discounted long-term predecessor models

So far, we derived the gradient of the logarithm of the stationary state distribution and showed it to be approximately proportional to the expected gradient of the log policy, evaluated at samples obtained from the long-term predecessor distribution $\mathcal{B}^{\pi_\theta}$. We now propose to train a model $\mathcal{B}^{\pi_\theta}_\omega$ to represent $\mathcal{B}^{\pi_\theta}$ and use its samples to estimate $\nabla_\theta \log d^{\pi_\theta}(\overline{s})$. However, rather than unrolling a time-reversed Markov model in time, which is prone to accumulated errors, we propose to use a generative model to directly generate jumpy predictions. We have furthermore found that imposing an order on autoregressive models achieves good results and propose to use a factored representation:

$$\mathcal{B}^{\pi_\theta}_\omega(s, a|\overline{s}) := \mathcal{B}^s_{\omega_s}(s|\overline{s}) \mathcal{B}^a_{\omega_a}(a|s, \overline{s}). \tag{7.6}$$

In this representation, $\mathcal{B}^s_{\omega_s}$ corresponds to a discounted long-term predecessor model over states while $\mathcal{B}^a_{\omega_a}$ corresponds to a probabilistic model of a goal-conditioned policy, similar to Pathak *et al.* [78]. In this work, both distributions will be modeled by Conditional Masked Autoregressive Flows [76] (see Chapter 6). To train this model, we collect training data using self-supervised roll-outs: We sample states, actions and target-states where the separation in time between the state and target-state is selected randomly based on the geometric distribution parameterized by γ as a training set for $\mathcal{B}^{\pi_\theta}_{\omega}$.

Training data for $\mathcal{B}^s_{\omega_s}, \mathcal{B}^a_{\omega_a}$ are obtained by executing the current policy to obtain a sequence $s_0, a_0, s_1, a_1, \cdots$. Next, to obtain a training sample, we first pick $s = s_t$ and $a = a_t$ for a random t. We now select a future state $\overline{s} = s_{t+j+1}$ from that sequence. For any particular s_{t+j+1} we now have

$$s, a \sim q^{\pi_\theta}_t(s_t = \cdot, a_t = \cdot | s_{t+j+1} = \overline{s}) \approx q^{\pi_\theta}(s_t = \cdot, a_t = \cdot | s_{t+j+1} = \overline{s}). \tag{7.7}$$

Note that in the episodic case, we can add transitions from terminal to initial states and pick t to be arbitrarily large such that the approximate equality becomes exact (as outlined in Section 2.1). In non-episodic domains, we find the approximation error to be small for most t. Finally, we choose j at random according to a geometric distribution $j \sim Geom(1-\gamma)$ and have a training triple $s, a, \overline{s}$ that can be used to train $\mathcal{B}^a_{\omega_a}$ and $\mathcal{B}^s_{\omega_s}$ as it obeys

$$s, a \sim (1-\gamma) \sum_{j=0}^{\infty} \gamma^j q^{\pi_\theta}_t(s_t = \cdot, a_t = \cdot | s_{t+j+1} = \overline{s}) = \mathcal{B}^{\pi_\theta}(\cdot, \cdot | \overline{s}). \tag{7.8}$$

7.2.3 GPRIL

To apply the gradient to imitation learning, we consider two scenarios: First, state-distribution matching by itself can be a useful imitation learning objective. In our evaluation (Section 7.3), we will focus on robotic manipulation task from proprioceptive features. In such tasks it is common that we record joint-positions as well as velocities which empirically

provides the agent with sufficient information to find good trajectories. The advantage of state-distribution matching is that demonstrations can be recorded without recording expert actions, e.g. via kinesthetic teaching. State-action distribution matching, however, is a more precise objective in that it unambiguously aims to recover the expert's policy. As in Chapter 5, we can achieve this by combing the state-distribution matching gradient with the behavioral cloning gradient:

$$\nabla_\theta \log \rho^{\pi_\theta}(\overline{s}, \overline{a}) = \nabla_\theta \log \pi_\theta(\overline{a}|\overline{s}) + \nabla_\theta \log d^{\pi_\theta}(\overline{s}). \tag{7.9}$$

where $\nabla_\theta \log \pi_\theta(\overline{a}|\overline{s})$ can be computed directly by taking the gradient of the policy using the demonstrated state-action pairs and $\nabla_\theta \log d^{\pi_\theta}(\overline{s})$ can be evaluated up to a constant factor using samples drawn from $\mathcal{B}_\omega^{\pi_\theta}(\cdot, \cdot|\overline{s})$ according to Equation 7.4. We arrive at the following estimate of the gradient:

$$\nabla_\theta \log \rho^{\pi_\theta}(\overline{s}, \overline{a}) \approx (1-\gamma) \log \pi_\theta(\overline{a}|\overline{s}) + \mathbb{E}_{s,a \sim \mathcal{B}^{\pi_\theta}(\cdot,\cdot|\overline{s})} \left[\nabla_\theta \log \pi_\theta(a|s) \right]. \tag{7.10}$$

This formulation of the state-distribution gradient yields an appealing practical point of view on the algorithm: Using the trained predecessor model, the agent imagines corrective samples. The imagined samples, together with the demonstration samples then serve as an augmented training set for Behavioral Cloning.

7.2.4 Practical Considerations

Finally, we will discuss a few practical implementation choices that help stabilize the training process. The full algorithm is described in Algorithm 3.

Using a replay buffer A primary focus of this chapter is to develop an imitation learning algorithm that is sample efficient. In practice, we find that training a long-term predecessor model requires comparatively few self-supervised samples; however, the number of

gradient descent steps required to train normalizing flows can be large and an on-policy approach would therefore not be sample-efficient. To alleviate this, we store self-supervised samples, collected asynchronously by an agent running in a separate process, in a short replay buffer. While our algorithm does not explicitly account for off-policy samples, we found empirically that a short replay buffer does not degrade final performance while significantly improving sample efficiency. Intuitively, we can expect a predecessor model which has been trained in this way to lag behind further than it normally would, similar to simply picking a lower learning rate.

Update schedules As is common for multi-stage optimization objectives in imitation- or reinforcement learning, we propose to train the predecessor model in parallel with the policy. We found, however, that simply performing a gradient descent step for both models can lead to instabilities and prevent the agent from learning a good policy. A possible source of these instabilities can be found in the self-reinforcing nature of the algorithm: if the policy is updated in an undesirable direction, the predecessor model will model this change and samples from the new policy will quickly be seen as more likely. The result is that imagined samples follow the "bad" policy and reinforce the undesired behavior. To alleviate this, we found it helpful to keep the predecessor model fixed for a longer period. We thus alternate between updating the predecessor model for $N_{\mathcal{B}}$ iterations, followed by updating the policy for N_π iterations.

Weighting distribution matching and behavioral cloning Equation 7.10 specifies a ratio between imagined samples and true samples that can be used to recover the gradient of the joint distribution; however, in practice we find that the behavioral cloning gradient is prone to overfitting and the agent may generalize better if a lower weight is applied to to the behavioral cloning gradient. We thus define a weighted version of the GPRIL gradient, where the base-case is given by $\beta_\pi = (1-\gamma); \beta_d = 1$, learning from observation can be recovered by setting $\beta_\pi = 0; \beta_d = 1$ and Behavioral Cloning can be recovered by setting

$\beta_\pi = 1; \beta_d = 0$:

$$\nabla_\theta \log \rho^{\pi_\theta}(\overline{s}, \overline{a}) \approx (1-\gamma) \log \beta_\pi \pi_\theta(\overline{a}|\overline{s}) + \beta_d \mathbb{E}_{s,a \sim \mathcal{B}^{\pi_\theta}(\cdot,\cdot|\overline{s})} \left[\nabla_\theta \log \pi_\theta(a|s) \right]. \tag{7.11}$$

Picking a discount factor A final parameter that requires special consideration in GPRIL is the discount factor γ. Compared to existing reinforcement and imitation approaches, the discount factor has a higher impact on the complexity of the learning problem. For high values of γ, the predecessor model has to model a large time-horizon and the variance of the training samples used to train the model explodes. To this end, we choose significantly lower values of γ, such as 0.9 or 0.7. Such low values work well if recovery to the next demonstrated state is possible in relatively few steps. Using the same example, in expectation, training samples for predecessor state-action pairs will, respectively, be 10 or 3.3 time steps in the past of the target state. Deviations that require a significantly longer sequence of actions to recover from would not have meaningful corresponding training samples in this scenario.

Policy initialization In our experiments, we initialize the policy with random weights; however, we found it helpful to ensure that the variance of the parameters on the output layer is comparatively small (e.g. using a factor of 0.01). While the predecessor model is able to learn well from random noise, strongly biased initial policies can lead to self reinforcing behavior and slow down the learning process significantly.

7.2.5 Unrolling the state-distribution gradient

Here, we derive Equation 7.3 which unrolls the recursive definition of $\nabla_\theta \log d^{\pi_\theta}(\overline{s})$ and rewrites it such that it can be replaced by an expectation over states and actions along

trajectories leading to the state $\overline{s}$. To do so, we first unroll said definition:

$$\begin{aligned}\nabla_\theta \log d^{\pi_\theta}(\overline{s}) &= \int q^{\pi_\theta}(s_t = s, a_t = a | s_{t+1} = \overline{s}) \left(\nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s)\right) ds, a \\ &= \lim_{T\to\infty} \int \Bigg(q^{\pi_\theta}(s_{t+T-1}, a_{t+T-1} | s_{t+T} = \overline{s}) \prod_{j=0}^{T-2} q^{\pi_\theta}(s_{t+j}, a_{t+j} | s_{t+j+1}) \\ &\qquad \sum_{j=0}^{T} \nabla_\theta \log \pi_\theta(a_{t+j}|s_{t+j}) \Bigg) ds_{t:t+T-1}, a_{t:t+T-1} + \\ &\qquad \int q^{\pi_\theta}(s_t = s, a_t = a | s_{t+T} = \overline{s}) \nabla_\theta \log d^{\pi_\theta}(s) ds, a\end{aligned} \tag{7.12}$$

Note that $\lim_{T\to\infty} q^{\pi_\theta}(s_t = s, a_t = a | s_{t+T} = \overline{s}) = \rho^{\pi_\theta}(s, a)$ due to Markov chain mixing and, therefore, the second term of the above sum reduces to 0 as

$$\int d^{\pi_\theta}(s) \pi(a|s) \nabla_\theta \log d^{\pi_\theta}(s) ds, a = 0. \tag{7.13}$$

By pulling out the sum, we can now marginalize out most variables and shift indices to arrive at the desired conclusion:

$$\begin{aligned}\nabla_\theta \log d^{\pi_\theta}(\overline{s}) &= \lim_{T\to\infty} \sum_{j=0}^{T} \int q^{\pi_\theta}(s_{t+j} = s, a_{t+j} = a | s_{t+T+1} = \overline{s}) \nabla_\theta \log \pi_\theta(a|s) ds, a \\ &= \lim_{T\to\infty} \sum_{j=0}^{T} \int q^{\pi_\theta}(s_t = s, a_t = a | s_{t+T+1-j} = \overline{s}) \nabla_\theta \log \pi_\theta(a|s) ds, a \\ &= \lim_{T\to\infty} \int \sum_{j=0}^{T} q^{\pi_\theta}(s_t = s, a_t = a | s_{t+j+1} = \overline{s}) \nabla_\theta \log \pi_\theta(a|s) ds, a\end{aligned} \tag{7.14}$$

7.2.6 GPRIL and the Policy Gradient Theorem

An alternative derivation of GPRIL can be found by connecting state-action-distribution matching to the policy gradient theorem [108]. The role of the policy gradient as $\gamma \to 1$ is to optimize the average reward $\sum_{s\in S} \rho^{\pi_\theta}(s, a) R(s, a)$. As such, we could directly derive the policy gradient in the average-reward case using the gradient of the state-action

distribution as has been done by Morimura et al. [69]. However, in this chapter we will instead rewrite the policy gradient theorem as presented in Sutton *et al.* [108] and use the alternative formulation to re-derive SAIL and GPRIL. By doing so, we can point out a direct connection between state-distribution matching and credit assignment and show that the discount factor γ used in both, SAIL and GPRIL, can be seen as identical to the discount factor used in reinforcement learning. The findings in this section are also crucial to relate GPRIL and VDI, the imitation learning approach we will derive in the following chapters. Note that for notational simplicity, we assume that the state space $\mathcal{S}$ and action space $\mathcal{A}$ are countable. Note that this is a common assumption made in the field [116] and considering that any state-action space is countable and finite when states and actions are represented using a finite number of bits makes it apparent that this assumption does not constitute a simplification in the practical sense.

The policy gradient, as defined in Sutton *et al.* [108], is optimizing expected reward by weighting the characteristic eligibility of each state-action pair that the agent encounters by its future long-term expected reward. In the following we will show that this forms a triangle sum which can be reversed, i.e. we will rewrite the policy gradient to sum over past characteristic eligibilities rather than future rewards. Following the notation by Sutton et al., we have

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s_j,a_j\sim\rho_\theta^\pi}\left[\nabla_\theta \log \pi_\theta(a_j|s_j)\mathbb{E}_{s_{j+1},a_{j+1},\dots}\left[\sum_{t=0}^{\infty}\gamma^t R(s_{t+j},a_{t+j})|\pi_\theta,s_j,a_j\right]\right] \tag{7.15}$$

We can now replace the expectation over the stationary distribution by an expectation over

the path of the agent. This yields a double sum whose order of summation can be changed:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \lim_{T\to\infty}\frac{1}{T}\mathbb{E}\left[\sum_{j=0}^{T}\nabla_\theta \log\pi_\theta(a_j|s_j)\sum_{t=0}^{T}\gamma^t R(s_{t+j},a_{t+j})|\pi_\theta\right] \\ &= \lim_{T\to\infty}\frac{1}{T}\mathbb{E}\left[\sum_{j=0}^{T}\sum_{t=j}^{T}\gamma^{t-j}\nabla_\theta \log\pi_\theta(a_j|s_j)R(s_t,a_t)|\pi_\theta\right] \\ &= \lim_{T\to\infty}\frac{1}{T}\mathbb{E}\left[\sum_{t=0}^{T}\sum_{j=0}^{t}\gamma^{t-j}\nabla_\theta \log\pi_\theta(a_j|s_j)R(s_t,a_t)|\pi_\theta\right]\end{aligned} \tag{7.16}$$

After changing the order of summation we can replace the outer sum with the expectation over the stationary distributions:

$$\nabla_\theta J(\pi_\theta) = \lim_{t\to\infty}\mathbb{E}\left[\sum_{j=0}^{t}\gamma^{t-j}\nabla_\theta \log\pi_\theta(a_j|s_j)R(s_t,a_t)|\pi_\theta\right] \tag{7.17}$$

This alternative form of the policy gradient theorem shows a more explicit form of credit assignment. Traditionally, solutions to the credit assignment problem focus on estimating the sum over future discounted rewards. Here, we can directly learn a model for past state-action pairs in order to directly reinforce it weighted by the reward, i.e. directly assign the "credit" to the correct state-action pairs. This gives rise to a policy gradient approach which uses a generative predecessor model rather than a learned value function which we will discuss further in the next section. The connection to imitation learning is already implied as the so found algorithm exhibits very similar steps to GPRIL, but we can make this connection more explicit by re-deriving the algorithm using Eq. 7.17

For imitation learning, we need to approximately find the gradient $\nabla_\theta \log\rho^{\pi_\theta}$ evaluated at demonstrated states and actions. Consider thus the gradient evaluated at $\overline{s},\overline{a}$:

$$\begin{aligned}\nabla_\theta \log\rho^{\pi_\theta}(\overline{s},\overline{a}) &= \frac{1}{\rho^{\pi}_{\theta}(\overline{s},\overline{a})}\nabla_\theta \sum_{s\in\mathcal{S},a\in\mathcal{A}}\rho^{\pi_\theta}(s,a)\mathbf{1}(s=\overline{s},a=\overline{a}) \\ &= \nabla_\theta \mathbb{E}_{s,a\sim\rho^{\pi}_{\theta}}\left[R(s,a)\right] =: \nabla_\theta J(\pi_\theta)\end{aligned} \tag{7.18}$$

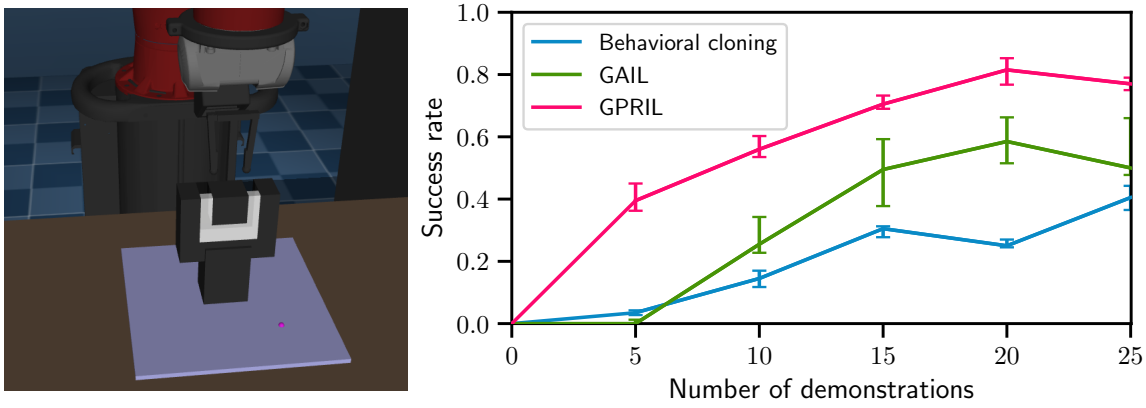
As we can see, the gradient is equivalent to the policy gradient using the reward function $R(s,a) := \frac{\mathbf{1}(s=\overline{s},a=\overline{a})}{\rho^{\pi}_{\hat{\theta}}(\overline{s},\overline{a})}$ in the average reward framework where $\hat{\theta}$ corresponds to the parameters of the policy at the current iteration. This reward function is not practical as it can be infinitely sparse and furthermore depends on the unknown stationary distribution. However, it allows us to derive GPRIL using the reversed PGT given in Eq. 7.17 as the discounted reward approximation of the above average-reward problem. Plugging in the definition of the reward, we have:

$$\begin{aligned} &\lim_{t\to\infty} \mathbb{E}\left[\sum_{j=0}^{t} \gamma^{t-j} \nabla_\theta \log \pi_\theta(a_j|s_j) R(s_t,a_t)|\pi_\theta\right] \\ &= \lim_{t\to\infty} \mathbb{E}\left[\sum_{j=0}^{t} \gamma^{t-j} \nabla_\theta \log \pi_\theta(a_j|s_j)|\pi_\theta, s_t=\overline{s}, a_t=\overline{a}\right] \end{aligned} \tag{7.19}$$

Finally, we notice that this equation constitutes the discounted version of Equation 7.3, thus we can immediately obtain our estimate of the state-action-distribution gradient:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1-\gamma} \mathbb{E}_{s,a\sim\mathcal{B}^{\pi_\theta}(\cdot,\cdot|\overline{s})}\left[\nabla_\theta \log \pi_\theta(a|s)\right] + \nabla_\theta \log \pi_\theta(\overline{a}|\overline{s}) \approx \nabla_\theta \log \rho^{\pi_\theta}(\overline{s},\overline{a}) \tag{7.20}$$

While this derivation is less direct than the derivation used in the previous sections, it draws a connection between the problem of matching state-action-distributions and the reinforcement learning problem with a reward that is positive for demonstrated state-action pairs and 0 otherwise. In the context of GPRIL, this indicates that the role of the discount factor is similar in both settings: Lower values of γ trade-off accurate matching of distributions for lower variance and as expedience. Agents with low γ will learn policies that recover and reach demonstrated states quicker over policies that are matching the experts state-action distribution more accurately long-term. Furthermore, this finding indicates possible steps to incorporate variance reduction techniques found in reinforcement learning in an approach like GPRIL. We will explore this further in the following chapters.



(a) (b)

Figure 7.1: **a)** Depiction of the clip-insertion task. **b)** Median final success rate and interquartile range out of 100 roll-outs over 8 seeds. GPRIL achieves the highest success rate, outperforming GAIL.

7.3 Evaluation

To evaluate our approach, we use a range of robotic insertion tasks similar to the domains introduced by Večerík *et al.* [121], but without access to a reward signal or, in some cases, expert actions. We choose these domains both for their practical use, and because they highlight challenges faced when applying imitation learning to the real world. Specifically, collecting experience using a robot arm is costly and demands efficient use of both demonstrations and autonomously gathered data. Furthermore, more difficult insertion tasks typically require complex searching behavior, particularly when the socket position is uncertain, and cannot be solved by open-loop tracking of a given demonstration trajectory. Insertion therefore poses a challenge for existing imitation learning and, especially, kinesthetic teaching approaches. We first compare against state-of-the-art imitation learning methods on a simulated clip insertion task, then explore the case of extremely sparse demonstrations on a simulated peg insertion task and finally, demonstrate real-world applicability on its physical counterpart.

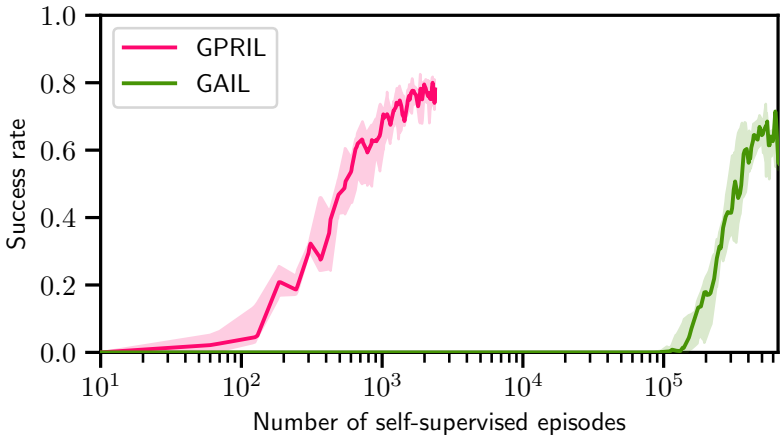


Figure 7.2: Median final success rate and IQR on clip insertion comparing sample efficiency. GPRIL is able to solve the task using several orders of magnitude fewer environment interactions.

7.3.1 Clip insertion

In the first task, a simulated robot arm has to insert an elastic clip into a plug which requires the robot to first flex the clip in order to be able to insert it (see Figure 7.1a). In real-world insertion tasks the pose of the robot, the socket, or the grasped object may be unknown. We capture this variability by mounting the socket on a pan-tilt unit which is randomized by ± 0.8 and ± 0.2 radians. To perform this behavior, the robot observes proprioceptive features, specifically joint position, velocity and torques as well as the position of the end-effector and the socket orientation as a unit quaternion. The task terminates when the robot leaves the work-space, reaches the goal, or after 50 seconds.

For comparative evaluation, we train a policy network to predict sufficient statistics of a multivariate normal distribution with independent dimensions over target velocities and train it using GPRIL, GAIL as well as behavioral cloning. We record expert demonstrations using tele-operation and normalize observations based on the recorded demonstrations. We then train GPRIL using a single asynchronous simulation and compare against the open source implementation of GAIL[1] for which we use 16 parallel simulations. We select the best hyper parameters found on a grid around the hyper-parameters used by Ho and Er-

[1] https://github.com/openai/baselines/tree/master/baselines/gail

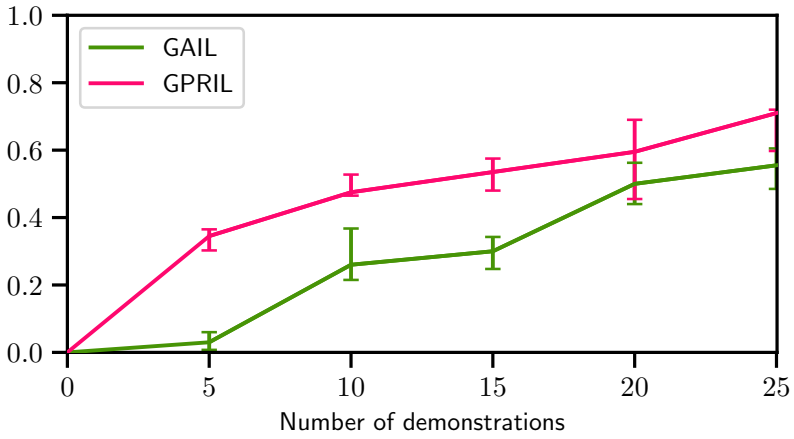


Figure 7.3: Comparison on clip insertion trained on states alone. Learning from states alone only slightly affects performance.

mon [39] but lower the batch size to 256 as it increases the learning speed and accounts for the significantly slower simulation of the task. We furthermore enable bootstrapping regardless of whether or not the episode terminated. As all discriminator rewards are positive, handling terminal transitions explicitly can induce a bias toward longer episodes. This is beneficial in the domains used by Ho and Ermon but harmful in domains such as ours where the task terminates on success. A detailed list of hyper-parameters can be found in appendix A.

We report final results after convergence and can see that both GAIL and GPRIL outperform behavioral cloning, indicating that generalizing over state-action trajectories requires fewer demonstrations than generalizing over actions alone. Furthermore, we observe a higher success rate using GPRIL and find that policies trained using GPRIL are more likely to retry insertion if the robot slides the clip past the insertion point. To compare sample efficiency of GPRIL to GAIL, we limit the rate at which the asynchronous actor is collecting data. While sample efficiency of GAIL could be increased by decreasing batch size or increasing various learning rates, we found that this can lead to unstable learning performance while reducing the amount of samples required by only a small amount. As can be seen in Figure 7.2, GPRIL requires several orders of magnitudes fewer environment interactions to learn this task. Finally, we evaluate the case where the expert's actions are unknown. Since

Figure 7.4: Depiction of the simulated and real peg-insertion task.

the state-space includes information about joint velocities as well as positions, we find that matching the state-distribution is sufficient to solve the task. GPRIL can achieve this by setting $\beta_d = 1$ and $\beta_\pi = 0$. As can be seen in Figure 7.3, performance deteriorates only marginally with a similar difference in performance between both methods.

7.3.2 Peg insertion with partial demonstrations

The second task is a simulated version of the peg-insertion task depicted in Figure 7.4. In this task, the robot has to insert the peg into the hole which is again mounted on a pan-tilt that randomly assumes pan and tilt angles varying by 0.4 and 0.1 respectively. Hyperparameters are largely identical and we report minor differences in Appendix A. Observation and action space are identical with the exception of the omission of torques from the observation space as they are not necessary to solve this task. We use this task

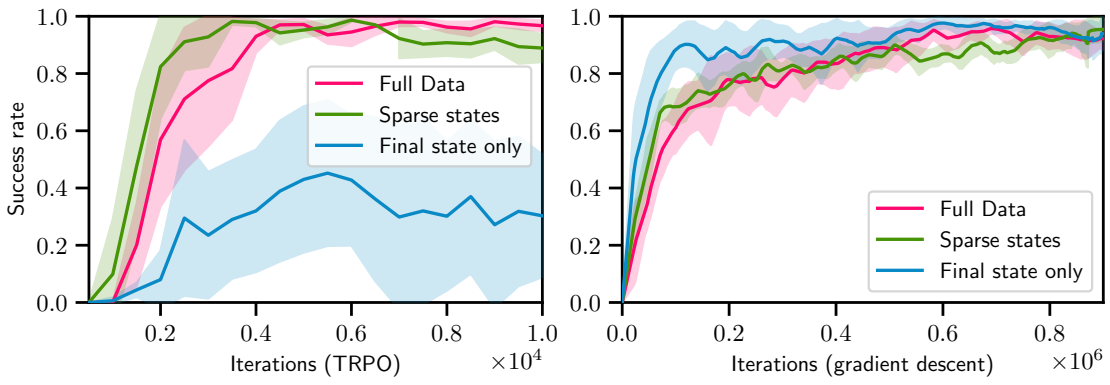
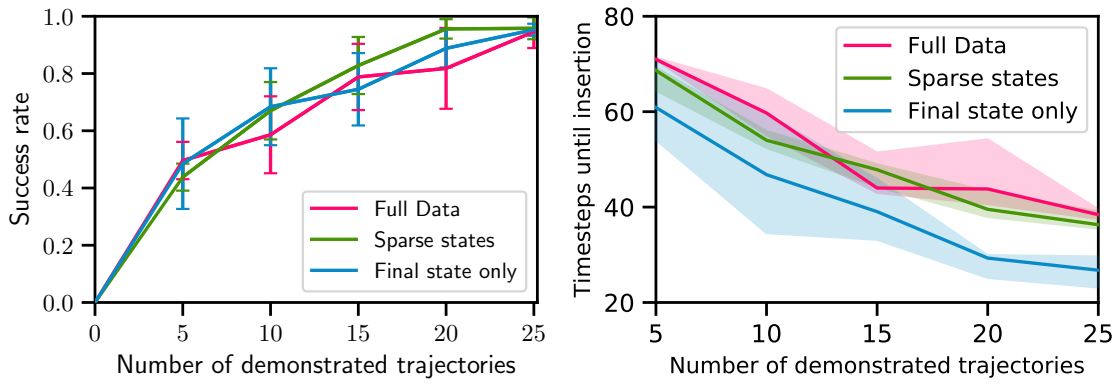


Figure 7.5: Average success rate and 95% confidence interval of **a)** GAIL with 25 demonstrations across 10 runs (evaluated over 100 roll-outs) and **b)** GPRIL across 5 seeds. Unlike GAIL, the performance of GPRIL doesn't drop off when provided with only final states.

to evaluate the performance of GAIL and GPRIL when learning from only a very limited set of demonstrated states. To this end, we compare three different scenarios in which the demonstrations are sparsified to varying degrees: In the first case, the agent has access to the full state-trajectories of the expert, in the second only every tenth state is available and in the third the agent sees only the final state of each of the 25 trajectories. Being able to learn from only partial demonstrations is a useful benchmark for the effectiveness of imitation learning methods but can also provide a convenient way of providing demonstrations and can free the agent to find more optimal trajectories between states. The setting we evaluate here is similar to using via-point demonstrations and shows that the approach developed here can be seen as a general, pure imitation learning alternative to the more specialized via-point-based approach presented in Chapter 4. As can be seen in Figures 7.6a and 7.6b, GPRIL achieves similar final success rates in all three scenarios while being able to learn a significantly faster insertion policy when learning from final states alone. We find that in the first two scenarios, this holds for GAIL as well (see Figure 7.5a), while in the third case, GAIL becomes highly unstable and the resulting performance can vary wildly, leading to a low average success rate. We hypothesize that these instabilities are a result of the discriminator overfitting to the very small amount of negative samples in its training data.



(a) (b)

Figure 7.6: Comparison of average success rate and insertion speed of GPRIL for differently sized demonstration sets. **a)** Average success rate and confidence interval of GPRIL. Final performance after 10^6 iterations increases steadily as the number of demonstrated trajectory increases but is unaffected by dropping steps from each demonstration. **b)** Median length and IRQ of trajectories that are successfully inserting the peg. Providing only final states is significantly faster.

7.3.3 Peg insertion on a physical system

In previous sections we demonstrated sample-efficiency that indicates applicability of GPRIL to real-world physical systems. To test this, we evaluate our approach on two variations of the physical peg-insertion task depicted in Figure 7.4 involving a physical pan-tilt unit which is fixed in one scenario and has pan and tilt angles varying by 0.1 and 0.02 radians in the second scenario. For each scenario we provide 20 demonstrations using kinesthetic teaching, which constitutes a natural way of recording demonstrations but provides state-trajectories only [17]. Hyper-parameters are altered from Section 7.3.2 to trade off a small margin of accuracy for higher learning speeds and are reported in Appendix A. Note, however, that tuning hyper-parameters precisely is very difficult on a physical system. As can be seen in Figure 7.7, GPRIL is able to learn a successful insertion policy that generalizes to unseen insertion angles using just a few hours of environment interactions[2]. We report best-seed performance as we observe a high amount of variability due to factors outside the agent's control, such as the pan-tilt unit not reporting accurate information after physical

[2]Total time to collect and train on 2000 roll-outs was 18.5 and 16.5 hours on the fixed and changing versions of the task respectively. However, GPRIL converged to good policies significantly sooner.

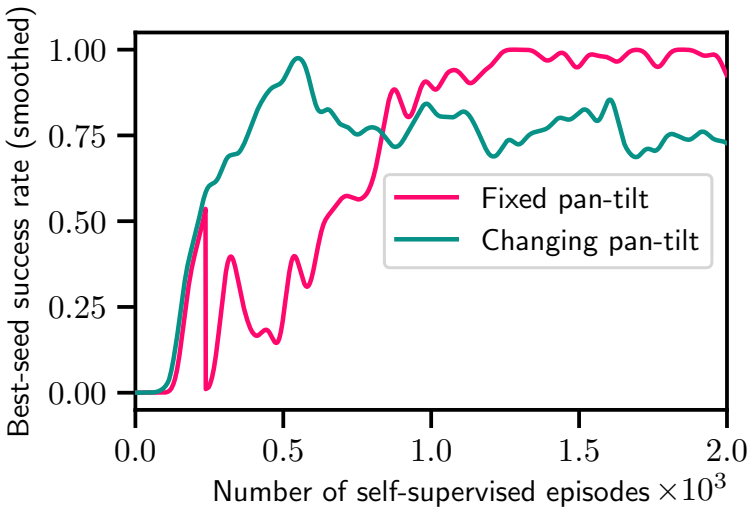


Figure 7.7: Best-seed performance of peg-insertion with the real robot.

contact with the robot. However, we wish to point out that the increased difficulty due to less predictable control is also likely to introduce additional variance that could be reduced further with careful design of exploration noise and other hyper-parameters. We furthermore provide a video of the training procedure and final policy to highlight the efficiency of our method[3] (see Figure 7.8 as well).

7.4 Summary

In this chapter, we introduced GPRIL, a novel algorithm for imitation learning which uses generative models to model multi-step predecessor distributions and to perform state-action distribution matching. We showed that the algorithm compares favorably with state-of-the-art imitation learning methods, achieving higher or equivalent performance while requiring several orders of magnitude fewer environment samples. Importantly, stability and sample-efficiency of GPRIL are sufficient to enable experiments on a real robot, which we demonstrated on a peg-insertion task with a variable-position socket. Furthermore, we showed that the proposed method enables learning of general policies from a very small number of partial demonstrations on a similar, but more complex variant of the domain used in our

[3] https://youtu.be/Dm0OCNujEmE

first work.

GPRIL achieves state-of-the-art results on domains where collecting environment samples is expensive but the required planning horizon of the agent is low. The manipulation tasks used in Section 7.3 exhibit complex dynamics around contact points, but are significantly simpler in large parts of the state space. In comparison, the common humanoid walking benchmark task [15] requires a larger horizon if the number of demonstrations is low. With a higher discount factor, the increased variance in training samples makes training a long-term predecessor model via maximum-likelihood difficult. To address this, we find that incorporating variance-reduction techniques commonly found in reinforcement learning, such as control variates in policy gradients and bootstrapping in temporal difference learning, with learning long-term models is essential. This will be the central topic for the remainder of this thesis.

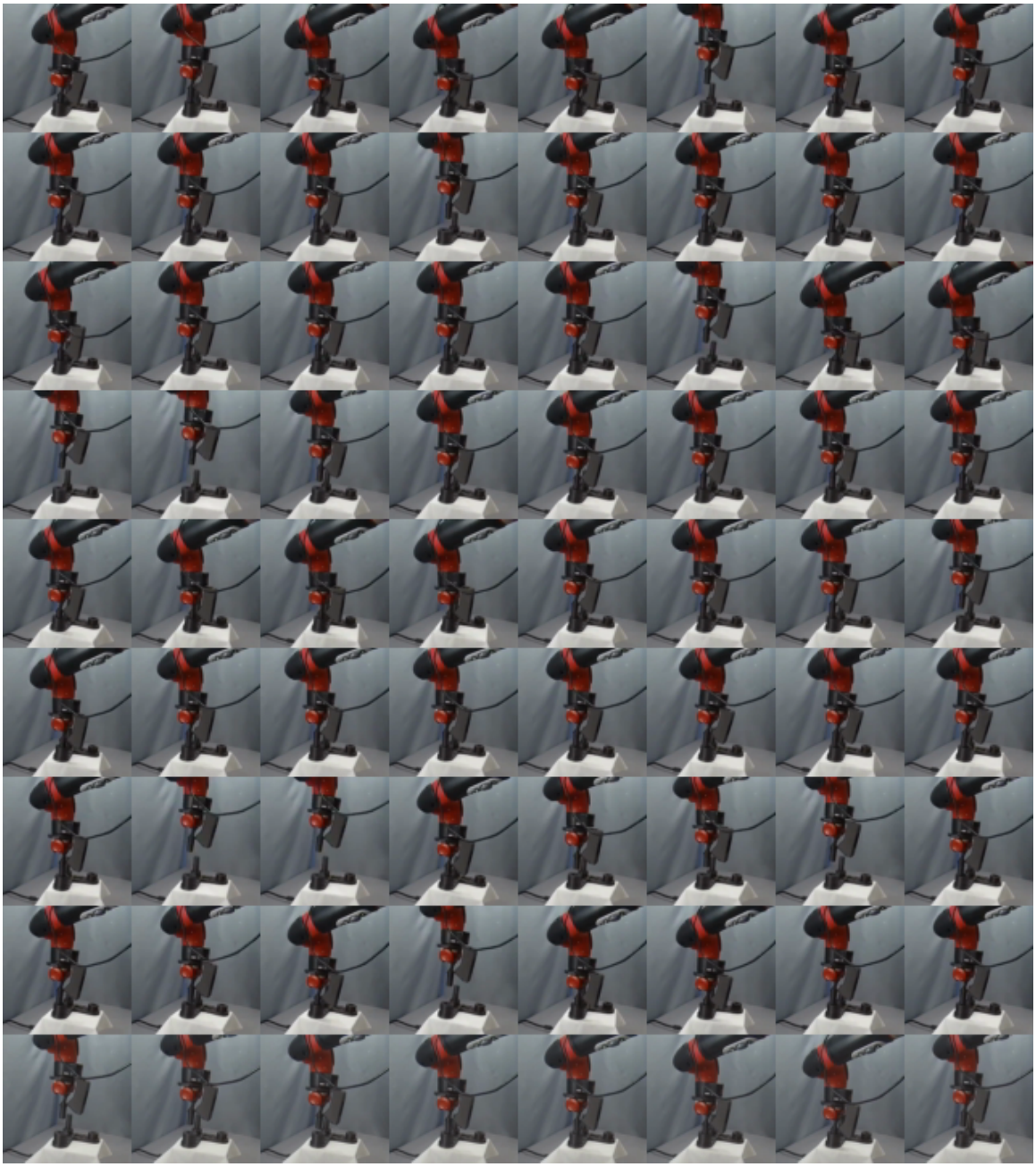


Figure 7.8: Animation of the training process on the real robot at a late stage

CHAPTER 8

UNIVERSAL VALUE DENSITY ESTIMATION

8.1 Motivation

SAIL and GPRIL show that effective imitation learning can be achieved by modifying the agent's state distribution. In both approaches, the agent is learning from self-supervised exploration how to get to the states that the expert has demonstrated and then uses that knowledge to act as the expert did. But the question of how to reach desired states is also within the purview of another, separate field of study: goal-conditioned reinforcement learning. Goal-conditioned reinforcement learning aims to train flexible agents that can solve multiple variations of a given task by parameterizing it with a goal. Often, this goal takes the form of a state or desired observation that the agent has to visit or achieve in the most optimal way. In this case, the task becomes highly similar to the setting of imitation learning: where SAIL and GPRIL learn to match the distribution of visited states to that of the expert, goal-conditioned reinforcement learning is typically concerned with a reproducing a single given state, i.e. the goal.

In the previous chapter, we have shown long-term generative models to be an efficient approach to imitation learning. Here, we show that a similar approach can be taken to address the problem of goal-conditioned reinforcement learning. Ultimately, we will show in Chapter 9 that this approach allows us to combine generative long-term models as used in GPRIL with a temporal difference approach, similar to SAIL, to arrive at an imitation-learning approach that is scalable, efficient in its use of demonstrations and able to solve tasks that require reasoning over large time horizons.

Despite significant achievements [94, 5, 72, 93], learning to achieve arbitrary goals remains an extremely difficult challenge. In the absence of a suitably shaped reward function,

the signal given to the agent can be as little as a constant reward give to the agent when the goal is achieved and when otherwise. Such a reward function is sparse and difficult to learn from, even if there is only a single goal and is prohibitively difficult to learn from when the task is to achieve any arbitrary goal. Hindsight Experience Replay (HER) [5] introduces the concept of hindsight samples as an effective heuristic to tackle this problem. Hindsight sampling selects transitions from past experience but artificially changes the desired goal to pretend that the agent intended to reach the state that it actually observed later in the roll-out. This way, the agent will frequently observe a reward and receive a comparatively dense learning signal. HER provides remarkable speed-ups and is capable of solving a variety of sparse, goal-conditioned RL problems; however, the approach cannot be applied to all domains as it suffers from hindsight bias: if an action has a large failure rate and only successfully leads the agent to the goal in a fraction of all attempts, the successful transitions will be the only ones used during training and the value of the action will be dramatically overestimated. For example, in a domain where the agent should avoid walking to close to a cliff for risk of falling, re-labeling trajectories as successful means that the agent will ignore all transitions where it attempted to reach a specific goal but fell down the cliff. This example is illustrated in Figure 8.1 and we analyze it further in Section 8.4.1.

Achieving unbiased goal-conditioned reinforcement learning with hindsight samples in the general case is impossible. If we never sample negative transitions, we cannot train the agent to accurately assess the risks of such transitions; however, the most common formulation, and a useful special case, assigns a positive reward to states in which the goal has been achieved and provides a reward of 0 otherwise[1] (possibly with an additional term handling action cost). We observe that the long-term expected reward in such a scenario is directly proportional to the discounted likelihood of achieving this goal. We propose the term "Value Density" to refer to a special case of a value function that is also a valid representation of this likelihood. We furthermore observe that hindsight samples provide

[1]This scenario is also the primary focus of the original Hindsight Experience Replay experiments

us with exactly the state-action and achieved goal triplets that are required for estimating the density of achieved goals. This gives rise to value density estimation: using a modern density estimator [21], we utilize hindsight samples to directly estimate the value function. Combined with regular temporal difference learning updates, this approach allows us to estimate the value function for each goal in a way that is sample-efficient, unbiased and low-variance. We will explore this approach in detail in Chapter 8.3.

8.2 Background

8.2.1 Goal-conditioned Reinforcement Learning

Goal-conditioned reinforcement learning trains an agent to solve many variations of the same task by conditioning the policy on a goal-vector g, i.e. learning a policies $\pi_{\theta;g}(a|s)$ or $\mu_{\theta;g}(s)$ based on reward functions $R(s, a; g)$. Work in the field focuses on efficient training of such goal-conditioned policies. A special case of goal-conditioned reinforcement learning attempts to achieve a particular state, or part of a state. For example, a robot arm may attempt to reach a particular position or move an object to a particular position. We can formalize this notion by defining a function $h(s, a)$ which indicates the "achieved" goal for each observed state-action pair and stating the reward as a function of the achieved goal and the desired goal $R\left(h(s, a); g\right)$.

Value functions are an important part of most approaches to model-free reinforcement learning and predict the long-term reward under a specific policy. To train a goal-conditioned policy, we can train a value function for each possible goal. The Horde architecture, proposed by Sutton *et al.* [109], first attempts to train multiple such value functions in parallel using recent developments in off-policy reinforcement learning and show that this is significantly more efficient than learning each objective individually. The value function can be seen as a projection of long-term dynamics based on a task-relevant measure, namely the reward. Limiting the information to be extracted based on the reward can be useful and often results in better policies when applied to problems with complex dynam-

ics; however, by learning multiple value-functions at once, Horde increases the amount of information extracted from each transition and is able to significantly increase the sample-efficiency of the agent. In Horde, the objectives are stated explicitly and separately and the agent maintains a separate model of the value for each reward function.

Building on Horde, Schaul *et al.* [94] introduce the concept of Universal Value Function Approximators and explore efficient representations as well as efficient training procedures for such UVFAs. A general value function generalizes the notion of a value function to be dependent on a given goal:

$$V^{\mu_{\theta;g}}(s;g) = E_{\mu_{\theta;g}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t; g)|s_0 = s\right]. \tag{8.1}$$

A UVFA aims to approximate all instantiations of a general value function in one representation. The training method which is most relevant for this paper is the straightforward direct bootstrapping approach proposed by Schaul *et al.* A goal is randomly sampled according to a given distribution and assigned to a transition sequence, the UVFA is then updated in an off-policy manner using a standard temporal difference update.

If the reward is sparse, for example if the reward is constant if g and $h(s, a)$ are close and 0 otherwise, training UVFAs can be difficult. In most transitions, the agent will not observe a reward and thus a large number of environment interactions are necessary to train a UVFA, if it is at all feasible. Hindsight updates (HER) [5] are a straight-forward solution, changing goals recorded in a replay memory based on what the agent has actually achieved by that transition. This ensures that each update is more likely to propagate non-zero value and enables remarkably fast learning in goal-conditioned domains with sparse rewards. However, it is important to note that HER learns a biased value function. We will explore the significance of this further in Section 8.3.1 where we also introduce an alternative solution that is both efficient and unbiased. To the best of our knowledge, the first paper to identify bias in HER is [57]; however, while Lanka and Wu propose a heuristic

Algorithm 4 Goal-conditioned RL via Universal Value Density Estimation

function UVD
  for $i \leftarrow 0..$#Iterations **do**
    Fill replay buffer with experience
    for s, a, g sampled from short replay buffer **do**
      Sample time offsets $t \sim \text{Geom}\,(1-\gamma)$
      Sample achieved goals g' t steps ahead of s
      Update F_ω with $-\nabla_\omega \log F_\omega\left(g'|s,a,g\right)$
    for s, a, s', g sampled from long replay buffer **do**
      $\overline{Q} \leftarrow \max\left(F_\omega\left(g|s', \mu_{\theta;g}\left(s';g\right), g\right), \gamma Q_{\overline{\phi}}\left(s', \mu_{\theta;g}\left(s'\right)\right)\right)$
      Update Q_ϕ with $\nabla_\phi \left(\overline{Q} - Q_\phi\left(s,a;g\right)\right)^2$
      Update $\mu_{\theta;g}$ with $\nabla_a Q_\phi\left(s,a;g\right)\Big|_{a=\mu_{\theta;g}(s)} \nabla_\theta \mu_{\theta;g}\left(s\right)$

method to handle this issue, no principled method to use hindsight samples in a completely unbiased way has been proposed to date.

Recent work by Nair *et al.* [72] and Sahni *et al.* [93] propose approaches that adapt HER to the setting of high-dimensional, visual features. While our work is based on RealNVP which was originally invented for modeling images, learning universal value densities in such a high-dimensional setting comes with its own set of challenges, not the least of which are computational, and is an interesting avenue for future research.

Most closely related to the method of Universal Value Density Estimation itself are Temporal Difference Models [85]. TDMs use a squared error function as a terminal value and use bootstrapping to propagate this value over a finite horizon. F_γ can similarly be seen as a model of terminal value, where the use of it as a lower bound allows bootstrapping with an infinite time-horizon. While there are further similarities to our method when using a Gaussian density estimator, TDMs do not address hindsight bias.

8.3 Approach

8.3.1 Addressing Hindsight Bias

At the heart of the problem of goal-conditioned reinforcement learning is the problem of learning a UVFA. Provided with an effective way to train a UVFA, a Q-learning or

deterministic policy gradient (DPG) based approach [103] can be taken to obtain a goal-conditioned policy with little difficulties. Before learning to model general value functions, we find it useful to first consider the case of learning a goal-conditioned value-function of a specific policy, i.e.

$$V^{\mu_\theta}(s; g) = E_{\mu_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t; g) | s_0 = s\right]. \tag{8.2}$$

While the general value function as defined in Eq. 8.1 models how good the agent is at achieving a goal if it tries to, this value function models how good a specific policy is at achieving a goal. In Section 8.3.3, we discuss how this can be extended to the fully goal-conditioned case, where the policy is conditioned on the goal as well.

Andrychowicz *et al.* [5] train UVFAs efficiently by shaping the distribution of goals selected for each temporal difference update. By sampling the goal and the state transition pair (s, s') from the same trajectory (hindsight sampling), HER provides an intuitive way to speed up the training of a UVFA; however, the approach suffers from hindsight bias. This bias is present, even if the policy is fixed and the goal-conditioning is applied to the reward only. To identify the source of hindsight bias, we can examine the effect of changing each sample distribution in the hindsight temporal difference update rule. Learning a goal-conditioned Q-function Q_ϕ of a deterministic policy μ_θ using an off-policy sample distribution $b(s, a)$ (e.g. induced by a replay buffer), the update rule looks as follows:

$$\begin{aligned}
&\phi^{(k+1)} \leftarrow \phi^{(k)} + \alpha \Delta\phi \\
&\Delta\phi := \int b(s, a) p(s'|s, a) p(g) \nabla_\phi Q_\phi(s, a; g) \delta ds, a, s', g \\
&\delta := R(s, a; g) + \gamma Q_{\phi^{(k)}}(s', \mu_\theta(s'); g) - Q_{\phi^{(k)}}(s, a; g).
\end{aligned} \tag{8.3}$$

If $b(s, a)$ differs from the on-policy distribution ρ^{μ_θ}, this update rule is biased in the same way an out of distribution regression update will be biased. While out-of-distribution sample bias suffices to prevent convergence in some cases [107], a powerful function approx-

imator can often overcome the sample bias. As a result, off-policy deep reinforcement learning approaches typically ignore this issue [71, 27] and learn effective policies with arbitrary choices of b. Similarly, if the distribution of goals $p(g)$ differs from the distribution at test-time, the only bias will be out-of-distribution bias as the regression target remains unchanged. However, $p(s'|s,a)$ cannot readily be replaced by an alternative distribution. Doing so would alter the regression target and therefore introduce significant bias that cannot be overcome by changing the model. The source of hindsight bias is that the next state s' and the goal g are not sampled independently. Sampling from $p(s'|s,a)p(g|s,a,s')$ is identical to sampling from $p(s'|s,a,g)p(g|s,a)$, i.e. hindsight sampling is equivalent to altering the transition dynamics such that they are more likely to lead to the goal. In the cliffwalk example, this means that transitions that are not leading to the goal, i.e. transitions terminating down the cliff, have a artificially low probability.

We will show empirical examples of the effects of hindsight bias in Section 8.4. To address hindsight bias, we use density estimation as opposed to regression and show that this yields an unbiased approach to learning from hindsight samples.

8.3.2 Universal Value Density Estimation

We consider the special case where the achieved goal can be extracted from a given state-action pair via a function $h(s,a)$ and the task is defined as matching the achieved goal to the desired goal g. This scenario is common in goal-conditioned RL and is the same scenario that has been evaluated by Andrychowicz *et al.* [5]. In discrete environments, we can define such tasks by a reward that is a positive constant if the goal is achieved and 0 otherwise, we define:

$$R(s,a;g) := (1-\gamma)\mathbb{1}(h(s,a)=g),$$

where h is a function that defines the achieved goal for any given state-action pair. In slight abuse of notation[2], we extend this definition to continuous environments as

$$R(s, a; g) = (1 - \gamma)\delta_{h(s,a),g}$$

We can now show that the Q-function of such tasks forms a valid density function. Specifically, we notice that the Q function is equivalent to the discounted probability density over goals that the agent is likely to achieve when following its policy, starting from the given state-action pair:

$$\begin{aligned} Q^{\mu_\theta}(s, a; g) &= \mathbb{E}_{\mu_\theta}\left[\sum_{j=0}^{\infty} \gamma^j R(s_j, a_j; g) | s_0 = s, a_0 = a\right] \\ &= (1 - \gamma) \sum_{j=0}^{\infty} \gamma^j \int p^{\mu_\theta}(s_{t+j} = s' | s_t = s, a_t = a)\delta_{h(s',\mu_\theta(s')),g} ds' \\ &= \int F_\gamma^{\mu_\theta}(s' | s, a)\delta_{h(s',\mu_\theta(s')),g} ds' \\ &=: F_\gamma^{\mu_\theta}(g | s, a), \end{aligned} \tag{8.4}$$

where we extend the notation of long-term predictive models to achieved goals.

It follows that we can learn an approximation of the universal Q-function by approximating the value density $F_\gamma^{\mu_\theta}$. This can be done using modern density estimators such as RealNVPs (see Section 6). To train the model, we gather training samples from a roll-out $s_0, a_0, s_1, a_1, \ldots$, collecting state-action pairs $s = s_t, a = a_t$ at random time-steps t as well as future achieved goals $g = h(s_{t+j}, a_{t+j}); j \sim Geom(1 - \gamma)$.

8.3.3 Goal-conditioned Reinforcement Learning

The result in Eq. 8.4 shows us how to estimate the goal-conditioned Q-function of a specific policy. This tells us how good a given policy is at achieving any possible goal and will be a

[2]Formally, the reward would have to be defined to be non-zero only in an ϵ-ball around $h(s, a)$ and inversely proportional to the volume of this ball. All results hold in the limit $\epsilon \to 0$.

useful result for imitation learning in the next chapter. For goal-conditioned reinforcement learning, we need to ask a different question: how good is the agent at achieving a given goal if it is specifically aiming for it. Notationally, the difference is subtle. Conditioning the policy on the desired goal g', we have:

$$Q^{\mu_{\theta;g'}}(s, a; g) = F_\gamma^{\mu_{\theta;g'}}(g|s, a), \tag{8.5}$$

As we aim to model all instantiations of $F_\gamma^{\mu_{\theta;g}}$ by a single parametric model F_ω, the model has to be conditioned on the intended goal g': $F_\omega(g|s, a, g')$. In the following, we therefore use $F_\omega(g|s, a, g')$ when our policy is goal-conditioned, and $F_\omega(g|s, a)$ otherwise. To train the model, we use achieved goals (hindsight samples) as before, but also record the intended goal (unaltered samples) and condition the model on it. To train the universal-Q-function efficiently, the agent needs to be able to learn from failures. If the agent achieves g' while attempting to achieve g, this should teach the agent about how to achieve g'. Here, we rely on the model's ability to generalize. Using density estimation, the agent is able to efficiently learn about its ability to reach g' while following a policy conditioned on g. When queried on the value of reaching goal g' while following a policy conditioned on g', the parametric model is able to utilize this experience to generalize. Like Hindsight Experience Replay, but unlike basic reinforcement learning algorithms that don't use hindsight samples at all, the algorithm will be able to learn a goal-conditioned solution even if the agent never achieves its intended goal during training. Like basic reinforcement learning, but unlike Hindsight Experience Replay, the algorithm is unbiased and finds optimal solutions even in stochastic domains.

8.3.4 Combining Temporal-Difference Learning and Value Density Estimation

Learning a model of $F_\gamma^{\mu_{\theta;g}}$ already provides us with a valid estimator of the Q-function; however, relying on density estimation alone is insufficient in practice. As the discount

factor approaches 1, the effective time-horizon becomes large. As a result, estimating Q purely based on density estimation would require state-action-goal triplets that are hundreds of time-steps apart, leading to updates which suffer from extremely high variance. Temporal difference learning uses bootstrapping to reduce the variance of the regression gradient and handle such large time-horizons. We propose to combine temporal difference learning and density estimation to derive an accurate estimator of Q that can handle sparse data.

There are a variety of ways in which the two estimators can be integrated. Here, we propose to use temporal difference learning as a lower bound. To justify this approach, we notice that there are two possible kinds of errors that can be present in the model for any state-action-goal triplet: it may be over-estimating the value or it may be under-estimating the value. If the model is over-estimating the value, it is likely to receive more training samples for that state-action pair in the future and correct itself in time; however, the training samples are unlikely to cover the state-action-goal space sufficiently. Thus, the model will assign high likelihood to small parts of the state-action-goal space and underestimate the likelihood in the remaining space. As a result, the model is more likely to under-estimate the value than to over-estimate it. Using a temporal-difference target as a lower bound on the value allows us to combat this source of error directly. As the reward is strictly positive, the value of each state-action pair is lower bounded by the value of the next state-action pair:

$$Q^{\mu_{\theta;g}}(s, a; g) \geq \mathbb{E}_{\mu_\theta}[\gamma Q^{\mu_{\theta;g}}(s', \mu_{\theta;g}(s'); g)]. \tag{8.6}$$

Furthermore, the reward is only non-zero if the goal is achieved. This means that in many practical domains, this lower bound is exact for large parts of the state-action space.

This leads us to the following temporal difference regression target:

$$\gamma \max(Q_\phi(s', \mu_{\theta;g}(s'); g), F_\omega(g|s', \mu_{\theta;g}(s'), g))) \tag{8.7}$$

where s, a, s', g are sampled as in Eq. 8.3. Note that this temporal-difference target can naturally be integrated in any reinforcement learning algorithm, allowing us to inherit the benefits of a modern approach such as TD3 [27]. The full algorithm is also outlined in Figure 4.

8.3.5 Practical considerations

There are a number of implementations decisions that were made to improve the sample-efficiency and stability of Value Density Estimation. Here, we review these decisions in more detail.

Using an exploration policy: In most cases, self-supervised roll-outs will require the agent to explore. In our method, we combine a temporal difference update rule as is usually found in deterministic policy gradients with density estimation. While the temporal-difference update rule can handle off-policy data from an exploration policy, density-estimation is on-policy. In practice, however, Fujimoto *et al.* [27] add Gaussian noise to the target-Q function and report better results by learning a smoothed Q-function that is akin to the on-policy Q-function with Gaussian exploration noise. It thus stands to reason that we can omit off-policy correction in the density-estimator.

Truncating the time horizon: The training data for learning a long-term model can be noisy. While we can expect density estimation to be efficient over a horizon of just a few time-steps, the variance increases dramatically as γ increases. This is the primary motivation for utilizing temporal-difference learning in conjunction with universal value density estimation. To better facilitate stable training, we truncate the time-horizon of the density-estimator to a fixed number of time-steps T. The temporal-difference learning component is thus solely responsible for propagating the value beyond this fixed horizon. The effect this has on the optimal policy is small: the Q-value will be underestimated by ignoring visitations with time-to-recurrence greater than T. A greater time-horizon boosts

the effect of hindsight samples and leads to a learning signal that is less sparse while a smaller time-horizon reduces the variance of the estimator.

Using a replay buffer: Using a replay buffer is essential for sample-efficient training with model-free reinforcement- and imitation-learning methods and improves the stability of deterministic policy gradients. We find that this is true for training long-term generative models as well. While importance sampling based off-policy correction for density estimation is possible, we find that the reduced effective sample size introduces instabilities and is thus undesirable. Instead, we propose to use a separate, shorter replay-buffer for density estimation to mitigate the undesirable effects of off-policy learning while retaining some of the benefits.

Normalizing states: As our method depends on density estimation, the resulting values are heavily affected by the scale of the features. We therefore normalize our data based on the range observed in random roll-outs.

8.4 Evaluation

8.4.1 Cliffwalk

We first illustrate the issues that arise with a biased hindsight update rule on a modified version of the classic cliffwalk domain used to illustrate the difference between off-policy and on-policy approaches. This will allow us to show the effect of hindsight bias in a controlled and small environment. We show the same effect in a more complex domain in Section 8.4.3. Traditionally, cliff-walk environments are used as a didactic aid to demonstrate that Q-learning tries to find the optimal path even if exploration noise makes this unsafe [107]. With hindsight bias we can observe a similar effect, as the agent always expects the most optimal outcome. However, unlike off-policy learning, this includes environment noise rather than just exploration noise and the resulting policy is thus not optimal.

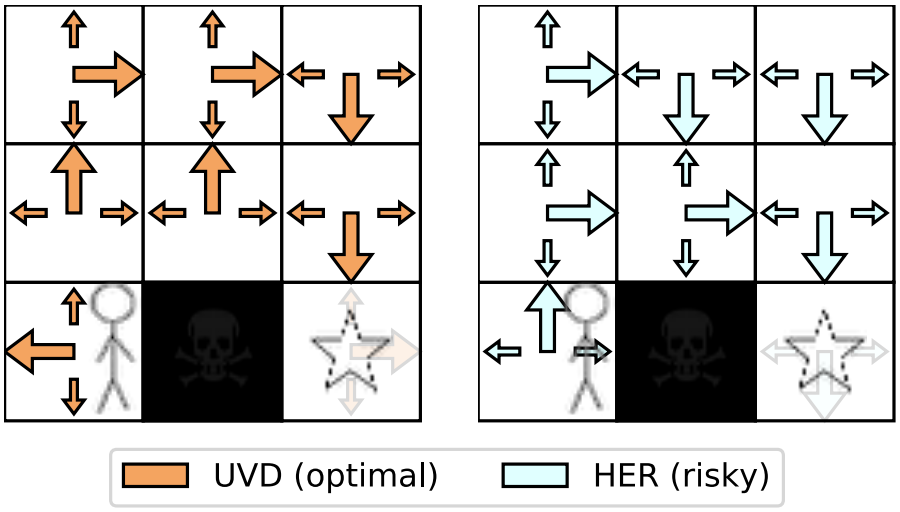
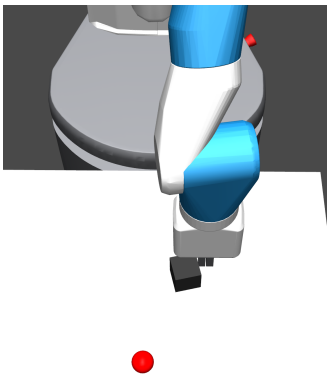


Figure 8.1: Simple cliffwalk domain. This domain exemplifies hindsight bias. Unlike the policy trained with UVD, the policy trained with HER chooses the shortest path and does not avoid the cliff.

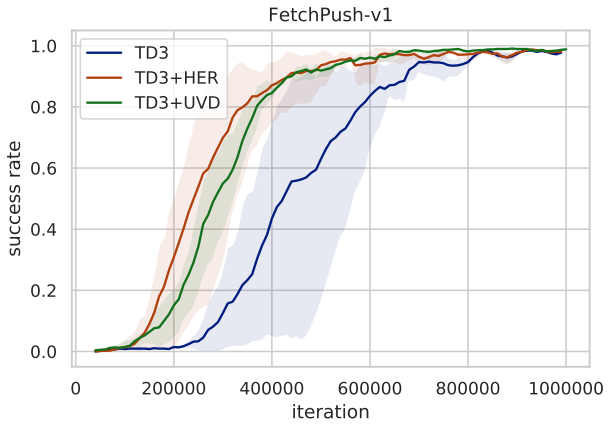
The domain is depicted in Figure 8.1. The agent starts in the bottom left corner and has to learn to reach any particular given goal; although we are primarily interested in the agent's ability to reach the bottom right corner. The bottom center tile is a pit (terminal state) and should be avoided. Large environment noise makes the agent move randomly with 50% probability, ensuring that the optimal strategy for the agent is to avoid standing next to the pit. If we train the agent with HER, we can see that the Q-function with respect to the goal is only updated if the agent did not fall into the pit. The natural consequence of this is that the agent overestimates the value of tiles next to the pit and thus chooses the shortest, sub-optimal route. The resulting policy can be seen in Figure 8.1. The UVD update rule takes a particularly simple form in the tabular setting; density estimation can be done simply by counting which goals are being reached. Using UVD, the agent converges to the optimal policy.

8.4.2 Deterministic Fetch Manipulation Tasks

The simulated manipulation suite involving a fetch robot was developed to be a standard benchmark for goal-conditioned reinforcement-learning with sparse reward signals and has



(a)



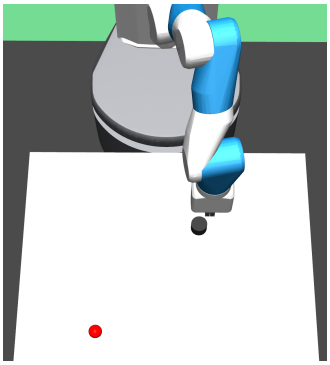
(b)

Figure 8.2: Comparison of TD3, UVD and HER on *FetchPush*.

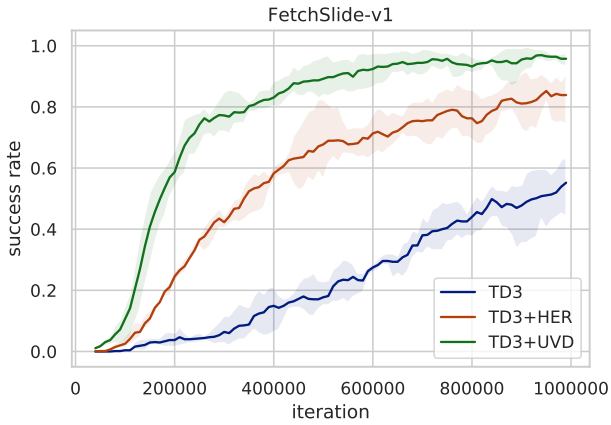
been introduced by Andrychowicz *et al.* [5] to show the strengths of hindsight experience replay. As the simulation is fully deterministic, the effects of hindsight bias don't play a role in these environments. Our goal here is therefore to show that Universal Value Density Estimation can solve the tasks as efficiently as HER. We compare all methods using the same hyper-parameters which can be found in Appendix B.

The first domain in the suite, *FetchPush*, requires the robot arm to learn to push an object to any given target location. The reward signal is sparse as the agent is given a non-zero reward only if the object reaches the desired location. In Figure 8.2, we can see that TD3 with HER and TD3 with UVD perform similarly. Contrary to the original findings by Andrychowicz *et al.* [5], we also find unmodified TD3 to be able to solve the task accurately using roughly twice the amount of training samples. This indicates that the area around the goal in which the object is considered to be at the desired location is relatively large. If we apply a stricter criterion for the goal being reached by reducing the size of the goal area by a factor of 100, we can see that hindsight samples are necessary to learn from sparse rewards. In this variant, TD3 fails to learn a useful policy while both UVD and HER performing similarly (see Figure 8.4a).

The second domain in the suite, *FetchSlide*, requires the robot to slide the object toward



(a)



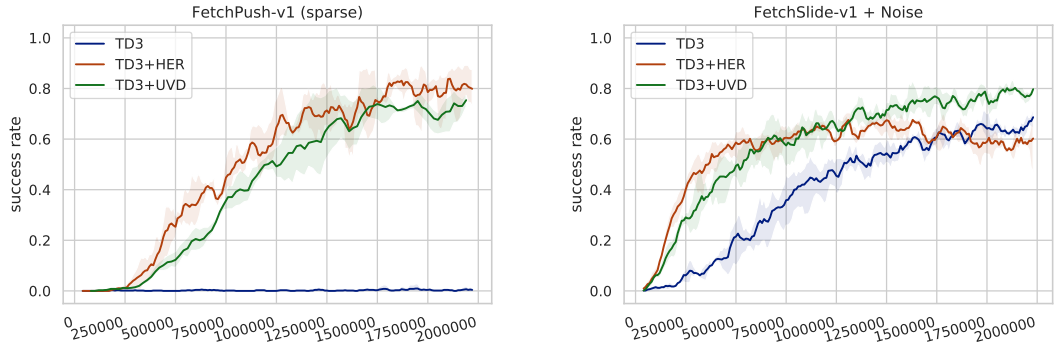
(b)

Figure 8.3: Comparison of TD3, UVD and HER on *FetchSlide*.

a desired location that is out of reach of the robot arm. In Figure 8.3, we can see UVD and HER learning to solve the task quickly while TD3 without hindsight samples requires significantly more training samples. Unlike in the case of *FetchPush*, TD3+UVD learns slightly faster in this domain than TD3+HER but both are able to solve the task eventually.

8.4.3 Noisy Fetch slide

Both, FetchSlide and FetchPush, are deterministic domains that play to the strengths of hindsight experience replay. In practice, however, manipulation with a real robot arm is always noisy. In some cases, HER can overcome this noise despite suffering from hindsight bias; however, this is not always the case. Here, we introduce a variation of the FetchSlide domain which presents difficulties to HER. We introduce Gaussian noise which is added to the actions of the agent. We scale this noise based on the squared norm of the chosen actions $\frac{1}{2e}||\max(0, a - 0.5 \cdot \mathbf{1})||_2^2$. This scaling allows the agent to move more slowly if the noise would make it impossible for it to solve the task optimally; however, doing so requires the agent to accurately assess the risks of its actions. In Figure 8.4b, we again compare TD3 without hindsight samples, TD3 with HER and TD3 with UVD. We can see that while HER initially learns quickly as in the deterministic domain, it converges



(a) (b)

Figure 8.4: Comparison of TD3, UVD and HER on variations of the Fetch manipulation domains. *FetchPush-Sparse* shows the advantage of hindsight updates when the required precision is significantly higher while *FetchSlide-Noisy* shows the effect of hindsight bias in stochastic domains.

to a policy that is noticeably worse than the policy found by TD3+UVD. This shows the effect of hindsight bias: in the presence of noise, the agent assumes the noise to be benign. Universal Value Density Estimation, on the other hand, estimates the risk accurately and achieves a higher success rate than HER and even TD3 without modifications is capable of solving this task and outperforming TD3+HER after receiving a large number of training samples.

8.4.4 Uniform Generalization

Finally, we consider an interesting variant of value density estimation for goal conditioned reinforcement learning. In Section 8.3.4, we introduced a form of the long-term predictive model $F_\gamma^{\mu_\theta;g}$ which is conditioned on the desired goal and showed it to be a Universal Value Function Approximator; however, conditioning on the desired goal also prevents us from using hindsight samples exclusively. Here, we analyze the performance of the agent if the unconditioned model $F_\gamma^{\mu_\theta}$ is used instead **(UVD-2)**. In this case, the learned Q-function will be biased: if a goal is more likely to be achieved by accident, i.e. if the goal is likely to be achieved while trying to achieve a different goal, the estimator will over-estimate the true Q-value. In practice, the likelihood of achieving a goal when intending to do so is

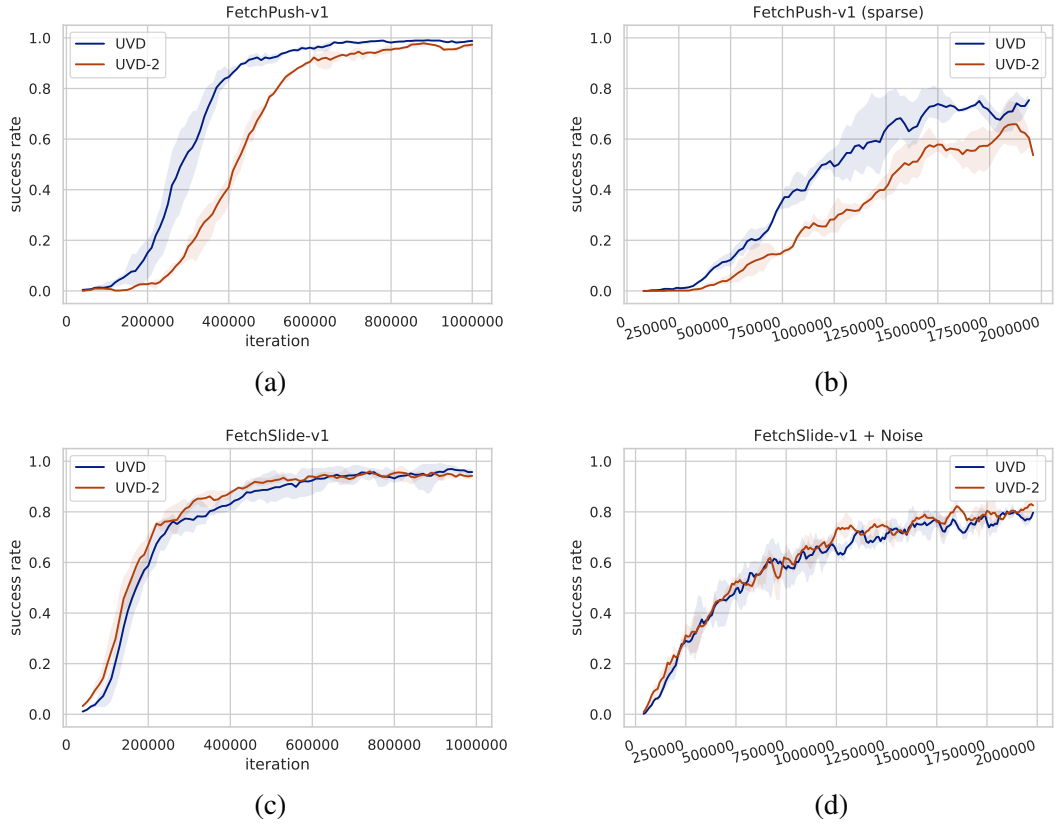


Figure 8.5: Comparison of UVD with and without conditioning on Fetch manipulation domains. The methods perform similar on the *FetchSlide-v1* task with the more correct, conditioned version showing higher precision on the *FetchPush-v1* tasks.

significantly larger and the effect of the bias is therefore limited. Using the unconditioned model, however, can allow us to benefit from hindsight samples even in tabular domains where generalization is not possible. In Figure 8.5, we can see this heuristic to work well. While the biased UVD-2 heuristic learns lightly slower on the *FetchPush* task and does not achieve the same level of performance on the sparse variant which requires high precision, it matches performance of the unbiased UVD on the *FetchSlide* variants and significantly outperforms TD3.

8.5 Summary

In this chapter, we introduced Universal Value Density Estimation, an unbiased approach to efficient goal-conditioned reinforcement learning. We showed that discounted long-

term predictive models can be combined with temporal-difference learning to efficiently estimate the value-function for a specific class of reward function. The density-estimation update rule allows for unbiased use of hindsight samples and thereby provides a dense, albeit high-variance approach to learning value functions. In contrast, a standard temporal-difference update provides a sparse, but low-variance learning signal. We demonstrated an approach that combines both update rules to get the best of both worlds and train a universal value function with a learning signal that is dense, but low-variance.

Learning how to reach a desired state is an essential building block of successful imitation learning approaches and the algorithm we presented lays the foundation for developing a novel imitation learning method in the following chapter. Using discounted long-term predictive models as an estimator for universal value functions connects this work to GPRIL. The combination with a temporal-difference update will allow us to derive a low-variance imitation learning method that will prove to be effective even when the dynamics are highly non-linear and the necessary planning horizon is large.

CHAPTER 9
VALUE DENSITY IMITATION

9.1 Motivation

Equipped with an effective approach to train the agent to reach any desired goal state, we now consider the application of this technique to imitation learning. In Chapters 5 and 7, we introduced two imitation learning approaches which followed a maximum-likelihood approach to state-distribution matching. We showed that by drawing the agent toward demonstrated states, it is able to learn a policy that is robust, using only a very small number of demonstrations. At its core, the problem of state-distribution matching is highly similar to the problem addressed in goal-conditioned reinforcement learning: where goal-conditioned reinforcement learning attempts to reach a single goal state, imitation learning approaches attempt to reach a set of demonstration states. It is thus no surprise that we can utilize techniques from goal-conditioned reinforcement learning to address the problem of state-distribution matching. In this chapter, we introduce Value Density Imitation (VDI), a principled approach that utilizes Value Density Estimation in order to draw the agent toward demonstrated states and match the expert's state-distribution

Besides the general connection between goal-conditioned reinforcement learning and imitation learning, Universal Value Density Estimation also exhibits a more specific and more direct connection to GPRIL. Where GPRIL learns a long-term discounted predecessor model to teach the agent how to reach demonstrated states, Universal Value Density Estimation learns its time-reversed equivalent, i.e. the long-term discounted predictive model. Both models are represented by similar models and trained using the same density estimation procedure with the same procedure for collecting training samples; however, by incorporating a temporal-difference lower-bound in Universal Value Density Estimation,

we were able to reduce the variance of the learning procedure and learn more complex long-term models. This chapter can be seen as the development of an approach which incorporates the same temporal-difference lower bound in a modified version of GPRIL to enable it to learn models with more complex dynamics over long time-horizons. We will show that this allows us to achieve state-of-the-art results on harder variants of common benchmark tasks in imitation learning. Naturally, there is a compromise: while the inclusion of temporal-difference learning allows the agent to learn more complex long-term models, the required amount of environment interactions will necessarily be comparable with other temporal-difference learning approaches.

As in SAIL and GPRIL, Value Density Imitation trains the agent to match the expert's state- or state-action distribution by drawing it toward demonstration states. Utilizing the same method as in the previous chapter, we use self-supervised roll-outs and a combination of density estimation and temporal-difference learning to train a model of the universal value density function. Sampling demonstration-states as goals, we can use this model to teach the agent to reproduce the demonstrated states. Unlike in the previous chapter, but as in SAIL and GPRIL, the policy is not goal-conditioned and depends only on the current state and action. To ensure that the agent attempts to visit all states equally often, i.e. to ensure that the agent matches the expert's state-distribution rather than sticking to a subset of demonstrated states, we have to pick demonstration states as goals with higher probability if the state is unlikely to be visited by the agent and vice versa. To this end, we maintain a model of the agent's state-distribution as well. In the following, we will derive this approach in detail.

9.2 Approach

9.2.1 Value Density Imitation

We now derive Value Density Imitation from first principles. The algorithm's goal is to train the agent to imitate an expert's policy using only a few demonstration samples from

Algorithm 5 Value Density Imitation

function VDI
  for $i \leftarrow 0..$#Iterations **do**
    Fill replay buffer with experience
    for s, a sampled from short replay buffer **do**
      Sample time offsets $t \sim \text{Geom}\,(1-\gamma)$
      Sample target states $\overline{s}$ t steps ahead of s
      Update F_ω with $-\nabla_\omega \log F_\omega\,(\overline{s}|s,a)$
      Update d_ω with $-\nabla_\omega \log d_\omega\,(s)$
    for $\overline{s}$ sampled uniformly from expert data, s, a, s' from replay buffer **do**
      $\overline{q} \leftarrow \max\left(F_\omega\,(\overline{s}|s,a)\,, \gamma Q_{\overline{\phi}}\,(s', \mu_\theta\,(s'; \overline{s}))\right)$
      Update q_ϕ with $\nabla_\phi\,(\overline{q} - Q_\phi\,(s,a;\overline{s}))^2$
    for $\overline{s}$ sampled from expert data with $p = \frac{1}{d_\omega(\overline{s})}$, s, a, s' from replay buffer **do**
      Update μ_θ with $\nabla_a Q_\phi\,(s,a;\overline{s})\Big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta\,(s)$

the expert. Building on our findings in developing SAIL and GPRIL (Chapters 5 and 7), we attempt to estimate the state-distribution gradient $\nabla_\theta \log d^{\mu_\theta}(\overline{s})$ from environment interaction. As we found in those chapters, evaluating and following this gradient at demonstration samples teaches the agent to match the expert's state-distribution via the maximum-likelihood principle. By itself, this objective is ambiguous and does not require the agent to recover the expert's policy; however, in practice the state-space is often designed in a way such that state-distribution matching can be an effective approach to imitation learning. We will evaluate state-distribution matching as imitation from observation in Section 9.3. To use the gradient estimate for state-action distribution matching, we previously combined the state-distribution gradient with the behavioral cloning gradient

$$\nabla_\theta \log \rho^{\pi_\theta}(\overline{s}, \overline{a}) = \nabla_\theta \log \pi_\theta(\overline{a}|\overline{s}) + \nabla_\theta \log d^{\pi_\theta}(\overline{s}),$$

where we assume a stochastic policy π_θ in place of a deterministic one. We found, however, that the behavioral cloning gradient can dominate a noisy estimate of the state-distribution gradient and lead to overfitting. Scaling the gradients mitigates this effect but requires careful tuning. Instead, we propose an alternative approach in this chapter: by augmenting

the state to include the previous action that lead to the state, the agent attempts to match the joint distribution of action and next state. While matching the action-next-state distribution is not the same as matching the state-action distribution, the latter is heavily implied by the former and the environment's dynamics.

If applied to a single state, following the state-distribution gradient maximizes the likelihood of being in that state and thus teaches the agent to go to that state. It is thus no surprise that it can be shown to be equivalent to the policy gradient for the right goal-conditioned reward function. Specifically, the state-distribution gradient is equivalent to the weighted policy gradient in the average-reward setting if the reward is as defined for goal-conditioned reinforcement learning ($R(s, a) = \delta_{s,\overline{s}}$). We have:

$$\begin{aligned} \nabla_\theta \log d^{\mu_\theta}(\overline{s}) &= \frac{\nabla_\theta d^{\mu_\theta}(\overline{s})}{d^{\mu_\theta}(\overline{s})} \\ &= \frac{\nabla_\theta \int d^{\mu_\theta}(s)\delta_{s,\overline{s}} ds}{d^{\mu_\theta}(\overline{s})} \\ &= \frac{\nabla_\theta J(\theta; \overline{s})}{d^{\mu_\theta}(\overline{s})} \end{aligned} \tag{9.1}$$

Intuitively, the policy gradient leads the agent toward a demonstration state $\overline{s}$, while the weight ensures that all demonstration states are visited with equal probability. This gives rise to Value Density Imitation (also see Algorithm 5):

1. Using self-supervised roll-outs, learn the goal-conditioned Q function as in Chapter 8

2. Using the same training samples used to train the conditional density estimator $F_\omega(\overline{s}|s, a)$, train an unconditional density estimator to model the agent's state-distribution: $d_\omega(s)$

3. Sample demonstration states as goals with probability proportional to $\frac{1}{d_\omega(\overline{s})}$

4. Use the learned Q function to estimate the policy gradient with the sampled demonstration states as goals. This estimate is equivalent to $\nabla_\theta \log d^{\mu_\theta}(\overline{s})$ up to a constant factor and allows us to match the expert's state-distribution.

9.2.2 Practical considerations

Normalizing states: As in the case of goal-conditioned reinforcement learning, Value Density Estimation requires the state-space to be normalized. In imitation learning, however, we have access to demonstration data to achieve better normalization. We collect roll-outs from random policies and normalize the data based on the range found in those random roll-outs combined with the demonstration data.

Delayed density updates: Temporal-difference learning with non-linear function approximation is notoriously unstable. To help stabilize it, a common [68, 27] trick is delay the update of the target network and allow the Q-function to perform multiple steps of regression toward a fixed target. Since we use the long-term predictive model F_ω to calculate the temporal-difference regression target, we apply the same trick here. We maintain a target network $F_{\overline{\omega}}$ which we set to be equal to the online density estimator F_ω after a fixed number of iterations. We use the same procedure to maintain a frozen target network of the unconditional state density estimator d_ω.

Averaging logits: While the dimensionality of the goal in goal-conditioned reinforcement learning is typically small, Value Density Imitation requires us to use the entire state as a goal. This, however, can be difficult if the number of features is large. If the state-features are independent, the density suffers from the curse of dimensionality as it is multiplicative and the Q-values will be either extremely large or extremely small. Even if the true density function does not exhibit this property, Normalizing Flows predict the density as a product of N predicted logits and prediction errors are therefore multiplicative. To combat this, we take the average of the predicted logits rather than the sum, effectively taking the N-th square root of the Q-function. We find that this approximation works well in practice and justify it further in Section 9.2.3.

Bounding weights on demonstrated states: In Value Density Imitation, we sub-sample demonstrations states proportional to $\frac{1}{d_\omega(\overline{s})}$ to ensure demonstration states to be visited with equal probability. In this formulation, demonstration states that are especially difficult to reach may be over-sampled by a large factor and destabilize the learning process. To counteract this, we put a bound on the weight of each demonstration state: for each batch, the weights are normalized and an upper bound is applied.

Spatial and temporal smoothing: We apply two kinds of smoothing to the learned UVFA to improve the stability of the learning algorithm. Spatial smoothing ensures that similar state-action pairs have similar value and is achieved by adding Gaussian noise to training samples of the target state when training the density estimator. Temporal smoothing ensures that the learned value does not have sudden spikes. Instead of using $F_\gamma(\overline{s})$ as a temporal difference regression target, we use a mixture of the density estimation and the temporal-difference lower bound. Using a temporal smoothing factor λ, the full temporal difference regression target is then

$$\begin{aligned} &\max(\gamma Q_\phi(s', \mu_\theta(s'); \overline{s}), \\ &\qquad \lambda\gamma Q_\phi(s', \mu_\theta(s'); \overline{s}) + (1-\lambda) F_\omega(\overline{s}|s', \mu_\theta(s'))) \end{aligned} \tag{9.2}$$

Bounding the temporal-difference gradient As with prior work [68], we found it helpful for value density imitation to use a hinge-loss instead of the mean-squared error to train the critic.

9.2.3 Escaping the curse of dimensionality

We now consider the properties of the universal value density estimator and propose a slight variation that is easier to handle numerically. To see the numerical challenge in using RealNVP density estimators as value functions, we take a look another look at a single bijector; here, the bijector $f_\omega(z) = (s_\omega(z), t_\omega(z))$ is predicting an affine transformation of

$z \sim p_z(\cdot) = \mathcal{N}(0, I)$ to $x \sim p_x(\cdot)$, i.e. $x_i = s_\omega(z)_i z_i + t_\omega(z)_i$. The density of x is then given as $p_x(x) = e^{\sum_{i=0}^{N} s_\omega^{-1}(x) - \left(\frac{x_i - t_\omega^{-1}(x)_i}{s_\omega^{-1}(x)_i}\right)^2 + \log \frac{1}{\sqrt{2\pi}}}$. It is readily apparent that this value can easily explode, especially when used as a target Q value in the mean-squared loss of a temporal difference update. To combat this, we propose to scale the logits with the dimensionality N, i.e. we use $p_x(x) = e^{\frac{1}{N}\left(\sum_{i=0}^{N} s_\omega^{-1}(x) - \left(\frac{x_i - t_\omega^{-1}(x)_i}{s_\omega^{-1}(x)_i}\right)^2 + \log \frac{1}{\sqrt{2\pi}}\right)}$. As this corresponds to only a constant factor on $\log F_\gamma$, the gradient-based density estimation is not affected. What remains to be shown is the following: first, we show that $J(\theta)^{\frac{1}{N}}$ can be used in place of $J(\theta)$ in both, goal-conditioned reinforcement learning and in imitation learning without changing the optimal policy; second, we justify using $Q^{\mu_\theta}(s, a; g)^{\frac{1}{N}}$ to approximate the modified policy gradient $\nabla_\theta J(\theta)^{\frac{1}{N}}$; and, finally, we show that $F_\gamma^{\mu_\theta}(g|s, a)^{\frac{1}{N}}$ can be used in place of $F_\gamma^{\mu_\theta}(g|s, a)$ to estimate $Q(s, a; g)^{\frac{1}{N}}$.

Using the scaled objective $J(\theta)^{\frac{1}{N}}$: Here, we have to consider two cases. In the case of goal-conditioned RL, we have to show that $\max_\theta J(\theta) = \max_\theta J(\theta)^{\frac{1}{N}}$. To this end, it is sufficient to note that the reward function is strictly non-negative and thus $(\cdot)^{\frac{1}{N}}$ is a monotonous function. In Section 9.2.1, we show that we can estimate the state-distribution gradient as $\nabla_\theta \log d^{\mu_\theta}(\overline{s}) = \nabla_\theta J(\theta)^{\frac{1}{N}}; R(s) = \frac{\delta_{s,\overline{s}}}{d^{\mu_\theta}(\overline{s})}$. To show that $J(\theta)^{\frac{1}{N}}$ can be used for imitation learning as well, we have to show that it can be used to estimate the gradient $\nabla_\theta \log d^{\mu_\theta}(\overline{s})$:

$$\begin{aligned}
\frac{1}{N}\nabla_\theta \log d^{\mu_\theta}(\overline{s}) &= \nabla_\theta \log d^{\mu_\theta}(\overline{s})^{\frac{1}{N}} \\
&= \frac{\nabla_\theta d^{\mu_\theta}(\overline{s})^{\frac{1}{N}}}{d^{\mu_\theta}(\overline{s})^{\frac{1}{N}}} \\
&= \frac{\nabla_\theta \int d^{\mu_\theta}(s)^{\frac{1}{N}} \delta_{s,\overline{s}} ds}{d^{\mu_\theta}(\overline{s})^{\frac{1}{N}}} \\
&= \nabla_\theta J(\theta)^{\frac{1}{N}}; R(s) = \frac{\delta_{s,\overline{s}}}{d^{\mu_\theta}(\overline{s})^{\frac{1}{N}}}
\end{aligned} \tag{9.3}$$

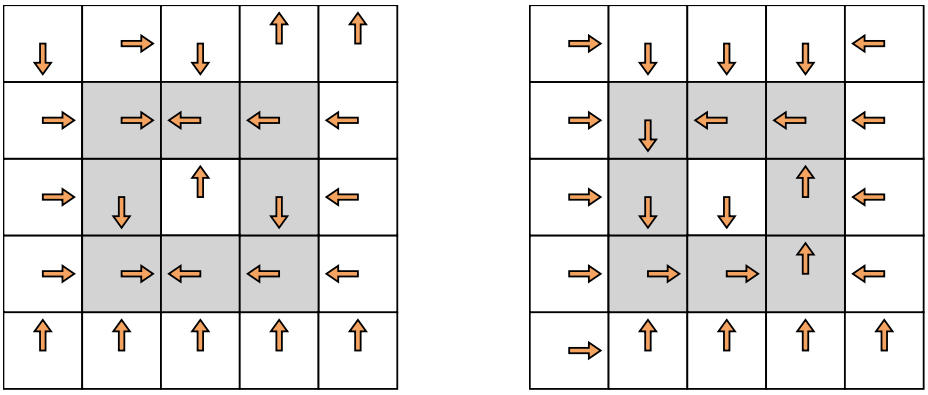
Estimating the policy gradient $\nabla_\theta J(\theta)^{\frac{1}{N}}$: The deterministic policy gradient theorem [103] shows that maximizing the Q-value in states sampled from the agent's discounted on-policy

state-distribution is equivalent to maximizing the reinforcement-learning objective. This is immediately apparent if the representation of the policy is sufficiently expressive and agent is able to take the action with maximum value in every state. Due to monotonicity of $(\cdot)^{\frac{1}{N}}$, this is true when using $Q(s,a;g)^{\frac{1}{N}}$ as well. If the policy is not able to maximize the Q-function everywhere, the deterministic policy gradient theorem shows that sampling from the discounted state-distribution leads to the agent making the right trade-offs. Using $Q(s,a;g)^{\frac{1}{N}}$, however, leads to a different trade-off. In practice, this is typically ignored: deterministic policy gradients used off-policy with a replay-buffer are not guaranteed to make the right trade-off and even if they are used on-policy, the discount-factor is typically ignored and the resulting estimate of the policy gradient is biased [74].

$F_\gamma(g|s,a)^{\frac{1}{N}}$ as TD target: In Section 8.3, we show that the Q function is equal to the density function F_γ: $Q(s,a;g) = F_\gamma(g|s,a)$. We then used the fact that the reward is non-negative to arrive at a lower temporal-difference bound $Q(s,a;g) \geq \mathbb{E}\left(\gamma Q(s', \mu_\theta(s'); g)\right)$. By using the larger of the two values as a regression target, we arrive at an estimator of Q that combines density estimation with temporal difference learning. As we are now estimating $Q(s,a;g)^{\frac{1}{N}} = F_\gamma(g|s,a)^{\frac{1}{N}}$, we now need to show a similar lower bound $Q(s,a;g)^{\frac{1}{N}} \geq \mathbb{E}\left[\gamma Q(s', \mu_\theta(s'); g)^{\frac{1}{N}}\right]$. To this end, it suffices to note that Q is strictly non-negative (since the reward is non-negative) and therefore $(\cdot)^{\frac{1}{N}}$ is a concave function. Then, we have:

$$\begin{aligned} Q(s,a;g)^{\frac{1}{N}} &= \mathbb{E}\left[R(s) + \gamma Q(s', \mu_\theta(s'); g)\right]^{\frac{1}{N}} \\ &\geq \mathbb{E}\left[\gamma Q(s', \mu_\theta(s'); g)\right]^{\frac{1}{N}} \\ &\geq \mathbb{E}\left[\gamma^{\frac{1}{N}} Q(s', \mu_\theta(s'); g)^{\frac{1}{N}}\right] \\ &\geq \mathbb{E}\left[\gamma Q(s', \mu_\theta(s'); g)^{\frac{1}{N}}\right] \end{aligned} \tag{9.4}$$

Note that while the last inequality holds, it indicates that we can achieve a tighter bound by using a higher discount factor $\gamma^{\frac{1}{N}}$.



(a) (b)

Figure 9.1: Comparison of distribution matching and support matching. **a)** shows learning with a positive reward assigned to demonstration states. **b)** shows distribution matching with full VDI, i.e. weighting by the current inverse likelihood of the agent visiting each demonstration state.

9.3 Evaluation

9.3.1 Circle world

First, we evaluate our algorithm on a small tabular domain. While this is a toy domain which does not itself pose a significant challenge, it allows us to demonstrate the efficacy of distribution matching, to validate VDI by showing its applicability to tabular domains and to contrast it with a more simple approach akin to support-matching approaches [e.g. 123]. To easily evaluate the effectiveness of distribution matching itself, we consider learning from observation alone, allowing us easily see deviations from the set of optimal policies.

The environment consists of a 5 by 5 gridworld in which the agent has to navigate with environment noise leading it to walk in a random direction with probability 0.5. The set of demonstrated states consist of the 8 states which constitute the inner circle of this gridworld. To evaluate state-distribution matching on this domain, we train an agent with value density estimation and compare it against an agent trained VDI without weighting of

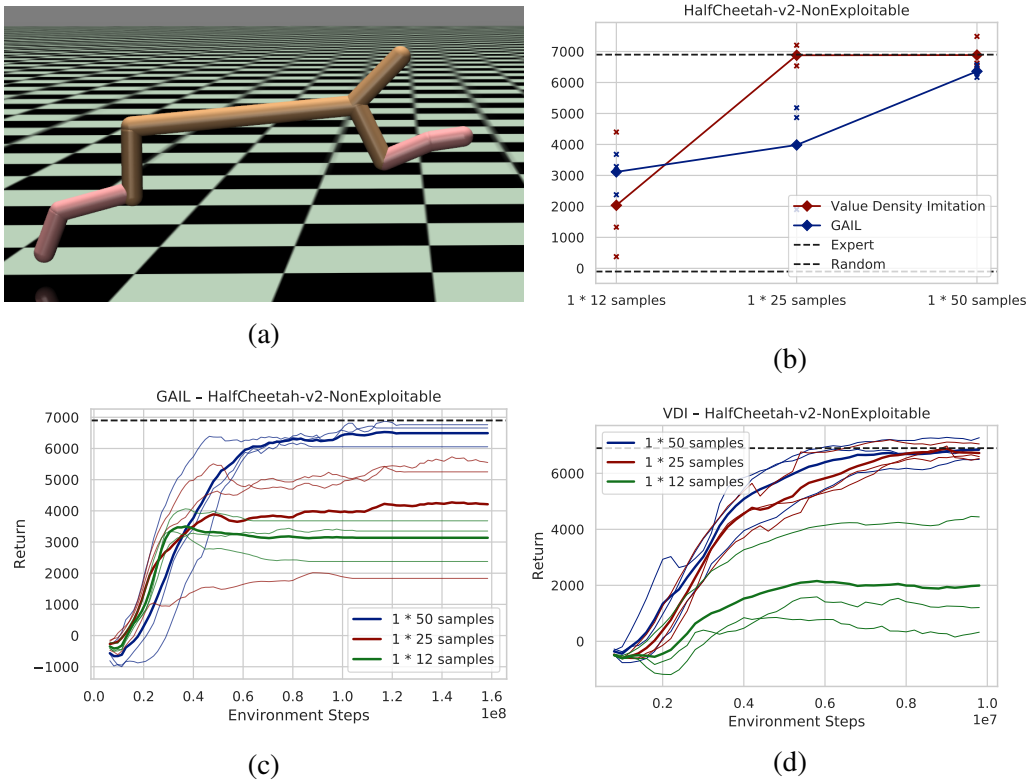
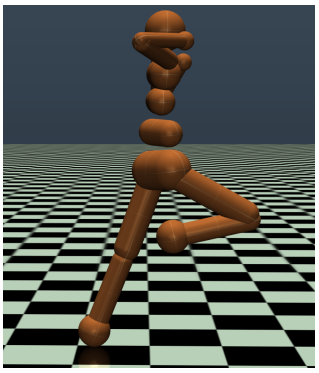
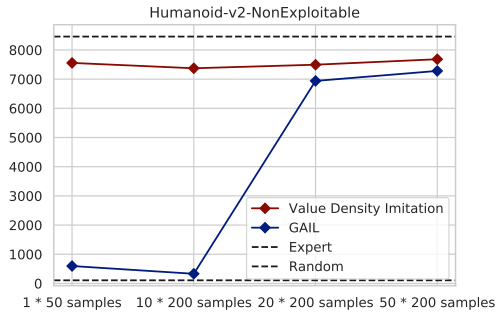


Figure 9.2: Comparison of GAIL and VDI on the HalfCheetah benchmark tasks. x-axis shows the number of provided demonstration trajectories as well as the number of state-action pairs collected from each trajectory. VDI is able to outperform GAIL when the number of provided demonstrations is low.

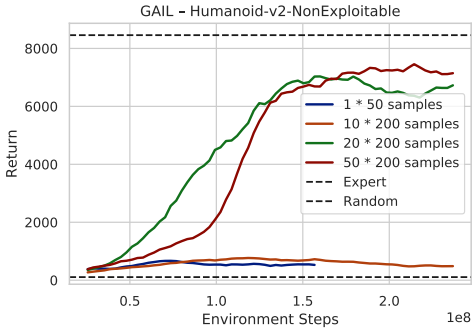
demonstrated states. The latter is equivalent to giving a reward of 1 on all demonstrated states and demonstrates the importance of matching the distribution accurately. There are two deterministic policies which match the distribution of states accurately, policies which lead the agent to walk in a circle in either direction, as well as a range of stochastic policies. In Figure 9.1, we can see that VDI converges to one of the deterministic policies. Provided only with a disordered set of states, the agent learns a policy leading it around in a circle. If the reward function is constant, however, the agent is drawn toward the circle but does not learn to complete the circle.



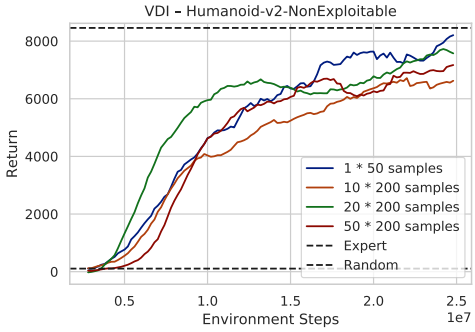
(a)



(b)



(c)



(d)

Figure 9.3: Comparison of GAIL and VDI on the Humanoid benchmark tasks. x-axis shows the number of provided demonstration trajectories as well as the number of state-action pairs collected from each trajectory. VDI is able to outperform GAIL when the number of provided demonstrations is low.

9.3.2 Locomotion domains

Next, we evaluate Value Density Imitation on common benchmark tasks and show that it outperforms the current state of the art. We point out issues with using these domains in an imitation learning context verbatim and introduce more difficult variations of the tasks that address those issues. The suite of simulated locomotion tasks found in OpenAI Gym [15] has become a standard benchmark task for reinforcement learning due to being physics based tasks with complex dynamics and proprioceptive, relatively low-dimensional state-action spaces. Recently, the suite of benchmark tasks has been used to evaluate imitation learning algorithms as well with Ho and Ermon [39] showing GAIL to solve the tasks using only a handful of demonstrated trajectories. Our goal is to show Value Density Imitation

to be able to achieve state-of-the-art demonstration efficiency; however, we find that the unmodified locomotion tasks, despite their popularity, are easily exploited in an imitation learning context and generally unsuitable for this kind of evaluation. While it is impressive that GAIL is able to solve *Humanoid-v2* with less than a dozen of disjointed demonstrated states, it is also clear that the agent is not learning to imitate the motion itself.

The primary source of reward in the locomotion task comes from the velocity in a particular direction; reward for survival has little influence on the optimal policy as falling leads to a velocity of 0 while the action cost in an optimal policy is almost completely dominated by the velocity reward. This is problematic as the velocity is fully observable as part of the state and, in the case of humanoid, may be encoded in more than one of the features found in the state-vector. Even the simple average of the state-features may be equivalent to a noisy version of the original reward signal. Moreover, as the reward is a linear combination of state-features, we know that accurate distribution matching is not necessary and matching feature expectations is sufficient [1, 39]. To alleviate this, we remove task-space velocities in x, y directions from the state-space. In most cases, this reduces the dimensionality of the state by 1 or 2. For humanoid locomotion it drastically reduces the dimensionality of the task. We find that this is sufficient for an agent to learn to solve the task and train expert policies using TD3 on the modified domains. A second source of bias can be found in the termination condition of the locomotion domains. Kostrikov *et al.* [52] point out that GAIL is biased toward longer trajectories and thus tries to avoid termination, which in the case of locomotion means to avoid falling. While Kostrikov *et al.* adjust the algorithm itself to avoid such bias, we instead propose to remove the termination condition and use an evaluation which cannot be exploited by a biased method. Beyond removing bias, this makes the learning problem significantly more difficult as the agent is allowed to diverge further from the demonstrated path and learning to recover from all states observed during training is more challenging.

Due to computational reasons, we focus on two locomotion tasks in particular: *HalfCheetah-*

v2 and *Humanoid-v2*. We choose *HalfCheetah-v2* as it is comparatively easy to train an agent to move in the right direction while it is comparatively difficult to move at high speeds. With the original threshold for solving the task being set at a score of 4500, recent advances in reinforcement learning train policies that achieve 3-4 times as high a score [27, 34] (although we find that removing velocity from the state reduces the top-speed achieved by the TD3-trained expert). Our second domain of choice is Humanoid-v2 which is generally considered to be the most complex locomotion task. Unlike in the *HalfCheetah-v2* domain, learning to move without falling can be a significant challenge for a learning agent. We furthermore find it sufficient to match state-distributions to solve *HalfCheetah-v2* and thus teach the agent from observation only when using VDI. In the case *Humanoid-v2*, we find that demonstrated actions significantly aid exploration and thus include them. We compare the demonstration-efficiency of Value Density Imitation with GAIL, the state-of-the-art in terms of demonstration-efficiency on these domains. Similar to Ho and Ermon [39], we subsample each trajectory to make the problem more challenging. The results can be seen in Figures 9.2 and 9.3. We find that the results reported by Ho and Ermon hold on the modified benchmark tasks, but can now also see the failure points of this baseline. While both methods are able to achieve near-expert performance on *HalfCheetah-v2-NonExploitable* using a single demonstrated trajectory sub-sampled at the same rate as used by Ho and Ermon, we find VDI to be able to imitate the expert even if the trajectory is sub-sampled at a lower rate. On *Humanoid-v2-NonExploitable*, we find the difference to be more drastic: while GAIL is able to learn locomotion behavior from a similar number of trajectories as used in the original paper (but sub-sampled at a greater rate), the performance drops off quickly if we reduce the number of trajectories further. VDI is able to imitate the expert given even a single trajectory.

Both methods are able to achieve great demonstration-efficiency by matching the expert's state-action distribution; however, GAIL does so by learning a distance function using demonstrations as training samples. As the number of demonstrations shrinks, learn-

ing a good discriminator to serve as a reward becomes more difficult, a problem that is especially apparent when the termination condition is removed and the majority of self-supervised state-action pairs are difficult to learn from. While it is possible that a discriminator could be trained in a way that still allows for it to be used as a good reward signal, VDI side-steps this issue altogether by learning a universal-Q-function independent from the demonstration data. The demonstrations are used to evaluate the Q-function and the small set of demonstrations is thus not used directly as training data for any network, preventing overfitting.

9.4 Summary

In this chapter, we introduced Value Density Imitation. Combining the findings of the previous chapters, VDI integrates temporal-difference learning with long-term models to achieve state-of-the art demonstration efficiency on more difficult variants of common benchmark tasks. By following a maximum-likelihood approach, VDI, like SAIL and GPRIL, avoids a common pitfall of imitation learning methods: the demonstration data is not used directly as supervised training data for any network. Where behavioral cloning uses the demonstration set as training data for the policy itself and adversarial approaches use the demonstration set as training data for a learned distance function, VDI uses the demonstrations for a monte-carlo sampling of goals. This way, Value Density Imitation is able to avoid overfitting at all levels and learn from demonstration sets that are extremely sparse.

Value Density Imitation is the last in a series of imitation learning approaches presented in this dissertation. Its reliance on model-free reinforcement learning means that improvements in sample-efficiency are necessary to match the efficiency of GPRIL, although, as a method that can be used in an off-policy setting, it requires orders of magnitudes fewer interactions than on-policy methods such as GAIL. As a distribution-matching method, it cannot utilize time information to the same effect as a trajectory based approach based on DMPs. As a method based on long-term models, it is not as truly model-free as SAIL.

Nevertheless, of all the methods we considered in this dissertation, it is the most widely applicable imitation learning method, able to handle the most complex dynamics with high stability and is generally the most powerful approach to learn to imitate the expert's policy from limited demonstration data.

CHAPTER 10
DISCUSSION

10.1 Contributions

Over the course of this dissertation, we introduced a series of imitation learning algorithms that allow us to train agents effectively with a very small number of demonstration samples. Starting with an algorithm to constrain a search over trajectories represented as Dynamic Movement Primitives, we showed that just a few demonstrated states can be sufficient to guide a reinforcement learning algorithm to a near-optimal solution. Lifting assumptions inherent in trajectory-based learning, we subsequently adopted a maximum-likelihood approach to match the expert's state distribution. We introduced SAIL, an algorithm that is able to learn robust policies from a single demonstrated trajectory without a reward signal. Following a temporal-difference learning approach to estimate the gradient of the state-distribution, it is the first algorithm out of three that enable us to guide the agent toward demonstrated states. The second approach, GPRIL, uses generative modeling techniques to learn a discounted long-term predecessor model and provides an alternative way to estimate the same gradient from samples of this model. The improved stability and sample-efficiency enable GPRIL to achieve state-of-the-art results on robotic manipulation tasks and allow it be applied directly to physical robot arms. Further improvements to the stability of the learning procedure allow Value Density Imitation to achieve state-of-the art results on common, highly non-linear benchmark tasks. Via the vehicle of universal value function approximation, VDI combines both approaches, long-term generative models and temporal-difference learning, and allows the agent to reason over long time horizons to reach the desired demonstrated state.

Connecting the algorithms is a more general approach expressed in the thesis state-

ment: **[A]Exploiting inherent structure in Markov chain stationary distributions allows learning agents to reason about likely future observations, and enables [C]robust and [D]efficient imitation learning, providing an [E]effective and interactive way to teach agents from [F]minimal demonstrations.**

To demonstrate this, we first set out to show that [F]reasoning over likely future observations enables the agent to learn from minimal demonstrations. A trajectory based approach gave us a way to investigate this aspect in an idealized scenario. Using only via-points, we show the approach to be able to utilize minmial demonstrations in robot manipulation tasks. With GPRIL, we revisit a similar task using a more general algorithm. We show the algorithm to be able to learn a general policy to solve robot manipulation tasks from final positions alone. Following similar principles, VDI exhibits greater stability in tasks with highly complex dynamics and exhibits state-of-the-art demonstration-efficiency on common benchmark tasks in the field. [E]Learning from sparse demonstration data lightens the burden on the expert teacher, enables learning on more complex tasks and can also enable more interactive learning procedures as demonstrated in Chapter 4.

[A] Starting with a trajectory-based representation, which includes a notion of future states the agent will visit, allowed us to manipulate the trajectory rather than learning which states the agent will actually visit. The latter, however, is necessary to train agents that use general policy representations, are able of solving general tasks and react to changes and dynamic obstacles in the environment. The first sentence of the thesis statement speaks to how such reasoning can be achieved. Exploiting inherent structure in the state-distribution, we developed a general approach of how to reason over true future states and laid the foundation for the following algorithms, starting with SAIL. SAIL exploits the recursive structure of the stationary state distribution to estimate its gradient via temporal-difference learning. This structure is inherent, and following the gradient allows the agent to manipulate likely future observations. [C]We show policies trained with SAIL to be robust and effective. Exploiting the same structure, GPRIL shares these traits and more. [D]We demon-

strate the approach to be sample-efficient in both demonstration- and expert data. Using the same principle and still estimating the gradient of the state-distribution, $^{\mathbf{A}}$VDI reasons over likely future observations in a more explicit way. Maintaining a model of likely long-term future observations, VDI connects the abstract principle with the model itself. The result is an imitation learning algorithm that is able to teach agents from minimal demonstrations when the long-term dynamics are highly complex.

In Chapter 3, we introduced a variety of imitation learning approaches, many of which use the notion of state-action-distribution matching to effectively imitate the expert and throughout this dissertation we compared our approaches empirically and quantitatively with some of these approaches. Qualitatively, what separates SAIL, GPRIL and VDI from common approaches found in the literature is the principle of state-action-distribution matching via maximum-likelihood. Recently, it has become popular to view the problem of imitation learning through the lens of divergence-minimization (e.g [29]). In this view, all of the approaches we introduced in this dissertation fall under the umbrella of minimizing the KL-divergence between the expert's and the agent's joint distribution; however, this view disregards the nature of the approximations that are necessary to minimize the desired divergence measure. While all three algorithms minimize the same divergence, the way in which this is achieved algorithmically is notably different; nevertheless, minimizing the KL divergence promotes certain properties that we believe to be beneficial to sample-efficient approximations. We can draw a connection to supervised learning. Here, the KL-divergence is significantly more important than other divergences as it forms the basis of maximum-likelihood. Its structure allows for easy approximation of the divergence, using a fixed training set as samples from the underlying environment-distribution. In this formulation, the only function that has to be evaluated and possibly differentiated is the target model itself. In imitation learning, approximations are inherently more complex and thus other formulations have been studied extensively; nevertheless, we believe that the same structure of the KL-divergence is responsible for the state-of-the-art demonstration-

efficiency achieved by the algorithms in this dissertation.

It was the intention behind our work to develop algorithms that promote imitation learning and make it a more widely applicable tool to solve real world problems. While this ultimate goal may require more work still, the work in this thesis augments the imitation learning toolbox with several approaches and ideas that are capable of achieving state-of-the-art results on real-world tasks such as robot manipulation as well as widely adopted synthetic benchmarks. We hope the ideas of this work will enable future developments in the field and ultimately fulfill the promise of capable, easy-to-teach, real-world agents.

10.2 Limitations and future work

To conclude, we take a look beyond the algorithms introduced in this dissertation to discuss outstanding challenges and possible avenues for future research. Some of these challenges are aspects of imitation learning that our algorithms address partially but can still be improved while other challenges go beyond the objective of our imitation algorithms and may use the algorithms as building blocks.

Simplicity Imitation learning promises easy and efficient training of an agent using demonstrations given to it by an expert. In practice, when the imitation learning problem is complex, the process involves careful tuning by the system designer for each domain. In part, this can be credited to the reliance of many imitation learning algorithms on reinforcement learning. Reinforcement learning algorithms are known to have a large number of hyper-parameters as well as implementation details and to be sensitive to the choice of many of them [43]. Inverse reinforcement learning algorithms, adversarial imitation learning methods as well as VDI further compound the issue by learning a reward function to be used with a reinforcement learning method. This introduces additional hyper-parameters, design decisions and additional complexity. GPRIL does not utilize a reinforcement learning approach, but nevertheless utilizes a multi-stage optimization problem with a large number

of hyper-parameters. We believe that building on the principles behind our imitation learning algorithms to develop methods which achieve similar results with reduced complexity would be a promising avenue of future research.

Sample-efficiency A core concern in the design of the algorithms presented in this dissertation as well as of the thesis statement, is to enable imitation learning from minimal amounts of demonstration data; however, sample-efficiency in terms of additional environment interactions is equally important. These goals are at odds: when demonstration data is abundant, behavioral cloning can be the most effective strategy and no further environment interactions are necessary. On the other hand, learning from minimal demonstrations requires the agent to learn about the environment and therefore requires exploration. In Chapter 7, we showed GPRIL to be orders of magnitude more sample-efficient than prior methods, in turn enabling training on a real robot. Achieving the same sample-efficiency on methods more suitable to complex sequential decision making tasks with a long time horizon such as VDI remains a challenge. To achieve this goal, we identify two avenues of future research. First, the development or utilization of more sample-efficient reinforcement learning methods would improve the sample-efficiency of imitation learning methods that rely on an underlying reinforcement learning algorithm. Using reinforcement learning as a component in an imitation learning system allows the agent to benefit from advances in reinforcement learning. GPRIL, however, achieves greater sample-efficiency without using a reinforcement learning algorithm. Further developments of such methods may lead to improvements in sample-efficiency across a larger range of environments. For example, the development of variance-reduction techniques for training long-term predecessor models may allow for GPRIL to be used in domains such as the locomotion suite while retaining its sample-efficiency.

Precision The ability to reproduce desired states with high precision is at the core of many effective imitation learning algorithms. It enables the agent to recover from devia-

tions as well as to track a desired trajectory. It is also the primary focus of the benchmarks in the popular locomotion suite [15]. In the majority of these tasks, a single demonstrated trajectory is sufficient to teach the agent if the agent is able to precisely repeat this reference behavior in the presence of noise. Precision has been a focus of both, our methods as well as the baselines we compared our methods to. Regardless, additional improvements are still possible and necessary.

Ability to generalize Besides precision, the second challenge a domain may pose is the ability to generalize to different situations. The majority of the domains in the locomotion benchmark suite require high precision but no generalization. The reacher domain is the opposite. Due to its simple dynamics, learning to control the arm to reach a particular point is relatively straight-forward; however, the agent needs to learn to generalize to different situations. As a goal-conditioned task, the reacher domain falls into a special sub-category of domains where parts of the state cannot be manipulated by the agent and ergodicity assumptions are violated. This is a problem for the evaluation and comparison of imitation learning algorithms that make this assumption (including GAIL and VDI) and the adoption of benchmark tasks which test a broader range of capabilities while retaining ergodicity would help the development of agents with greater generalization capabilities. Such domains demand the ability of the agent to generalize to unseen situations but give the agent the ability to fully control its own state (for example manipulation tasks in which the robot has to manipulate objects with varying initial positions). Combining our methods with mechanisms that enable the agent to generalize more explicitly [e.g. 124, 20] would also be an interesting avenue for future research.

Combination with other objectives In Chapter 4, we introduced a combined reinforcement and imitation learning system. While the use of a reinforcement learning algorithm in this method was necessary to disambiguate between possible candidate trajectories that adhere to the demonstrated via-points, it also enabled the agent to incorporate a reward signal

to find an optimal solution while the demonstration signal allowed for rapid exploration. Combining reinforcement- and imitation learning objectives into a single learning system is not new with prior solutions using imitation learning as initialization [83], for exploration [37, 121] or as a second objective in a multi-objective optimization problem [127]; nevertheless, the optimal combination of the two learning signals remains an outstanding challenge. In this dissertation, we showed both, GPRIL and VDI to have algorithmic connections to reinforcement learning (see Section 7.2.6 and Chapter 8) which may enable a more principled approach to trade off both objectives and facilitate the development of an effective combined learning system.

Appendices

APPENDIX A

HYPERPARAMETERS FOR CHAPTER 7

General parameters	
Total iterations	$2e6$
Batch size	256
γ	0.9
Replay memory size	50000
$N_{\mathcal{B}}$	15000
N_{π}	5000
$\mathcal{B}^s$ **and** $\mathcal{B}^a$	
Stacked autoencoders	2
Hidden layers	500, 500
Optimizer	Adam
Learning rate	$2 \cdot 10^5$
Burnin	50000 iterations
L2-regularization	10^{-2}
$min(\sigma_i)$	0.1
Gradient clip, L_2 norm	100
Policy π_θ	
Hidden layers	300, 200
Optimizer	Adam
Learning rate	10^4
σ bounds	(0.01, 0.1)

(a) GPRIL parameters for clip insertion

General parameters	
Total iterations	$1e4$
#Processes	16
Batch size	$16 \cdot 256$
Actor steps per iteration	3
Discriminator steps per iteration	1
γ	0.995
Actor	
Hidden layers	300, 200
KL step size	0.01
σ bounds	(0.01, 0.1)
Discriminator	
Hidden layers	150, 100
Optimizer	Adam
Learning rate	10^4
Entropy regularization	1
Optimizer	Adam
Critic	
Hidden layers	300, 200
Optimizer	Adam
Learning rate	$5 \cdot 10^3$

(b) GAIL parameters for clip insertion

General parameters	
Total iterations	$5e6$
Batch size	256
Hidden layers	300, 200
σ bounds	(0.01, 0.1)
Optimizer	Adam
Learning rate	10^4
L2-regularization	10^{-4}

(c) BC parameters for clip insertion

	Simulation	Real robot
General parameters		
Batch size	256	
γ	0.9	0.7
Replay memory size	10000	50000
$N_{\mathcal{B}}$	10000	
N_{π}	1000	5000
$\mathcal{B}^s$ **and** $\mathcal{B}^a$		
Stacked autoencoders	2	
Hidden layers	500, 500	
Optimizer	Adam	
Learning rate	10^{-5}	$3 \cdot 10^{-5}$
Burnin	0	
L2-regularization	10^{-2}	10^{-3}
$min(\sigma_i)$	0.1	0.01
Gradient clip (L_2)	100	
Policy π_θ		
Hidden layers	300, 200	
Optimizer	Adam	
Learning rate	10^{-4}	
σ bounds	(0.01, 0.1)	

(d) GPRIL parameters for peg insertion

General parameters	
#Processes	16
Batch size	$16 \cdot 256$
Actor steps/iteration	3
Discriminator steps/iteration	1
γ	0.995
Actor	
Hidden layers	300, 200
KL step size	0.01
σ bounds	(0.01, 0.1)
Discriminator	
Hidden layers	150, 100
Optimizer	Adam
Learning rate	10^{-4}
Entropy regularization	1
Optimizer	Adam
Critic	
Hidden layers	300, 200
Optimizer	Adam
Learning rate	$5 \cdot 10^{-3}$

(e) GAIL parameters for simulated peg insertion

APPENDIX B

HYPERPARAMETERS FOR CHAPTER 8

General parameters	
Environment steps per iteration	1
γ	0.98
Batch size	512
Replay memory size	1500000
Short replay memory size	50000
Sequence Truncation (Density estimation training)	4
Optimizer	Adam
Policy	
Hidden layers	400, 400
Hidden activation	leaky relu
Output activation	tanh
Exploration noise σ	0.1
Target action noise σ	0.0
Learning rate	$2e-4$ (TD3, TD3+HER), $8e-4$ (TD3+UVD)
Q-network	
Hidden layers	400, 400
Hidden activation	leaky relu
Output activation	50 * tanh (TD3, TD3+HER), linear (TD3+UVD)
Learning rate	$2e-4$ (TD3, TD3+HER), $8e-4$ (TD3+UVD)
RealNVP	
Bijector hidden layers	300, 300
Hidden activation	leaky relu
Output activation, scale	tanh(log())
Output activation, translate	linear
num bijectors	5 (slide), 6 (push)
Learning rate	$2e-4$

Table B.1: Common parameters in Fetch environments

Here, we list the hyper-parameters used for TD3, TD3+HER and TD3+UVD on the Fetch experiments. When applicable, we largely use the same hyper-parameters for each algorithm. Two important exceptions are the learning rate which is tuned individually for each algorithm (TD3+UVD benefits reliably from higher learning rates whereas TD3 and TD3+HER does not always converge to a good solution at higher learning rates) as well as

the output activation of the Q-network. A tanh activation is used to scale the value to the range of -50 to 50 in the case of TD3 and TD3+HER as we found this to drastically improve performance. Since the density is not bounde from above, the same activation cannot be used in the case of TD3+UVD.

APPENDIX C

HYPERPARAMETERS FOR CHAPTER 9

Parameter	*HalfCheetah*	*Humanoid*
General parameters		
Environment steps per iteration	10	10
γ	0.995	0.995
Batch size	256	256
Replay memory size	1500000	1500000
Short replay memory size	500000	500000
Sequence Truncation (Density estimation training)	4	4
Optimizer	Adam	Adam
Policy		
Hidden layers	400, 300	400, 300
Hidden activation	leaky relu	leaky relu
Output activation	tanh	tanh
Exploration noise σ	0.3 (until iteration 25000), 0.1 (after)	0.1
Target action noise σ	0.0	0.0
Learning rate	$3e-4$	$3e-4$
Q-network		
Hidden layers	400, 400	400, 400
Hidden activation	leaky relu	leaky relu
Output activation	linear	linear
Learning rate	$3e-4$	$1e-4$
RealNVP		
Bijector hidden layers	400, 400	400, 400
Hidden activation	leaky relu	leaky relu
Output activation, scale	tanh(log())	tanh(log())
Output activation, translate	linear	linear
num bijectors	5	5
Learning rate	$1e-4$	$2e-5$
L2-regularization	$1e-5$	$1e-6$
Spatial smoothing	0.1	0.1
Temporal smoothing	0.	0.98

Table C.1: VDI parameters in locomotion environments

Here, we list the hyper-parameters used for VDI in Chapter 9. Parameters are largely identical between environments; however, in some cases we trade off higher learning speed for reduced accuracy on *HalfCheetah*.

In the case of GAIL, we use the implementation found in OpenAI baselines[1], using 16 parallel processes (collecting 16 trajectories per iteration) and do not modify the parameters.

[1] https://github.com/openai/baselines/tree/master/baselines/gail

REFERENCES

[1] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine Learning*, R Greiner and D Schuurmans, Eds., ACM, 2004, pp. 1–8.

[2] B Akgun, M Cakmak, K Jiang, and A. Thomaz, "Keyframe-based learning from demonstration," *International Journal of Social Robotics*, no. 4.4, pp. 343–355, 2012.

[3] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz, "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective," in *International Conference on Human-Robot Interaction*, 2012, pp. 391–398, ISBN: 978-1-4503-1063-5.

[4] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, and Others, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[5] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[6] B. Argall, E. Sauser, and A. Billard, "Policy adaptation through tactile correction," in *Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, 2010.

[7] S. Arora, A. Risteski, and Y. Zhang, "Do GANs learn the distribution? some theory and empirics," in *International Conference on Learning Representations*, 2018.

[8] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching youtube," in *Advances in Neural Information Processing Systems*, 2018, pp. 2930–2941.

[9] M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine, "Stochastic variational video prediction," in *International Conference on Learning Representations*, 2018.

[10] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. TB, A. Muldal, N. Heess, and T. Lillicrap, "Distributional policy gradients," in *International Conference on Learning Representations*, 2018.

[11] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. Citeseer, 2017, vol. 1.

[12] S. Bitzer, M. Howard, and S. Vijayakumar, "Using dimensionality reduction to exploit constraints in reinforcement learning," in *Intelligent Robots and Systems*, 2010, pp. 3219–3225.

[13] A. Boularias, J. Kober, and J. Peters, "Relative Entropy Inverse Reinforcement Learning," in *International Conference on Artificial Intelligence and Statistics*, vol. 15, 2011.

[14] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *International Conference on Learning Representations*, 2019.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *OpenAI Gym*. 2016.

[16] S. Calinon and A. G. Billard, "What is the teacher's role in robot programming by demonstration?: Toward benchmarks for improved learning," *Interaction Studies*, vol. 8, no. 3, pp. 441–464, 2007.

[17] S. Chernova and A. L. Thomaz, "Robot learning from human teachers," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.

[18] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," in *Advances in Neural Information Processing Systems*, 2018, pp. 5032–5043.

[19] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine Learning Journal*, vol. 75, no. 3, pp. 297–325, 2009.

[20] Y. Ding, C. Florensa, P. Abbeel, and M. Phielipp, "Goal-conditioned imitation learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 298–15 309.

[21] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using Real NVP," *arXiv preprint arXiv:1605.08803*, 2016.

[22] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *Advances in neural information processing systems*, 2017, pp. 1087–1098.

[23] A. D. Edwards, L. Downs, and J. C. Davidson, "Forward-Backward Reinforcement Learning," *arXiv preprint arXiv:1803.10227*, 2018.

[24] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, and Others, "IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures," in *International Conference on Machine Learning*, 2018, pp. 1406–1415.

[25] C. Finn, S. Levine, and P. Abbeel, "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization," *International Conference on Machine Learning*, 2016.

[26] J. Fu, K. Luo, and S. Levine, "Learning Robust Rewards with Adverserial Inverse Reinforcement Learning," Feb. 2018.

[27] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018, pp. 1582–1591.

[28] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "Made: Masked autoencoder for distribution estimation," in *International Conference on Machine Learning*, 2015, pp. 881–889.

[29] S. K. S. Ghasemipour, R. Zemel, and S. Gu, "A divergence minimization perspective on imitation learning methods," in *Conference on Robot Learning*, 2019.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[31] A. Goyal, P. Brakel, W. Fedus, S. Singhal, T. Lillicrap, S. Levine, H. Larochelle, and Y. Bengio, "Recall Traces: Backtracking Models for Efficient Reinforcement Learning," in *International Conference on Learning Representations*, 2019.

[32] K. Gregor, G. Papamakarios, F. Besse, L. Buesing, and T. Weber, "Temporal Difference Variational Auto-Encoder," in *International Conference on Learning Representations*, 2019.

[33] S. Griffith, K. Subramanian, J. Scholz, C. Isbell, and A. L. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2625–2633.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, 2018, pp. 1856–1865.

[35] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *International Conference on Evolutionary Computation*, 1996, pp. 312–317.

[36] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2018.

[37] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstrations," in *AAAI Conference on Artificial Intelligence*, 2018.

[38] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "Beta-vae: Learning basic visual concepts with a constrained variational framework.," in *International Conference on Learning Representations*.

[39] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[40] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, "Neural autoregressive flows," in *International Conference on Machine Learning*, 2018, pp. 2083–2092.

[41] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[42] A. J. A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems*, 2002, pp. 1547–1554.

[43] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *arXiv preprint arXiv:1708.04133*, 2017.

[44] K. Judah, S. Roy, A. Fern, and T. G. Dietterich, "Reinforcement Learning Via Practice and Critique Advice.," in *AAAI Conference on Artificial Intelligence*, 2010.

[45] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos, "Recurrent Experience Replay in Distributed Reinforcement Learning," 2018.

[46] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *Ppt*, 2013.

[47] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 215–10 224.

[48] W. B. Knox and P. Stone, "Reinforcement learning from simultaneous human and MDP reward categories and subject descriptors," in *Autonomous Agents and Multiagent Systems*, 2012, pp. 475–482.

[49] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[50] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems*, 2008, pp. 849–856.

[51] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with EM-based reinforcement learning," in *Intelligent Robots and Systems*, 2010, pp. 3232–3237.

[52] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, "Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning," 2018.

[53] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 330–345, 2012.

[54] A. Kumar, S. A. Eslami, D. Rezende, M. Garnelo, F. Viola, E. Lockhart, and M. Shanahan, "Consistent jumpy predictions for videos and scenes," 2018.

[55] M. Kumar, M. Babaeizadeh, D. Erhan, C. Finn, S. Levine, L. Dinh, and D. Kingma, "Videoflow: A flow-based generative model for video," *arXiv preprint arXiv:1903.01434*, 2019.

[56] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann, "Data-Efficient Generalization of Robot Skills with Contextual Policy Search," in *AAAI Conference on Artificial Intelligence*, 2013.

[57] S. Lanka and T. Wu, "ARCHER: Aggressive Rewards to Counter bias in Hindsight Experience Replay," *arXiv preprint arXiv:1809.02070*, 2018.

[58] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine, "Stochastic adversarial video prediction," *arXiv preprint arXiv:1804.01523*, 2018.

[59] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[60] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems*, 2017, pp. 3812–3822.

[61] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," in *Advances in Neural Information Processing Systems*, 2018, pp. 700–709.

[62] S. Mannor, R. Y. Rubinstein, and Y. Gat, "The cross entropy method for fast policy search," in *Proceedings of the 20th International Conference on Machine Learning*, 2003, pp. 512–519.

[63] J. Merel, A. Ahuja, V. Pham, S. Tunyasuvunakool, S. Liu, D. Tirumala, N. Heess, and G. Wayne, "Hierarchical visuomotor control of humanoids," in *International Conference on Learning Representations*, 2019.

[64] J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess, "Neural probabilistic motor primitives for humanoid control," in *International Conference on Learning Representations*, 2019.

[65] J. Merel, Y. Tassa, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. Heess, "Learning human behaviors from motion capture by adversarial imitation," *arXiv preprint arXiv:1707.02201*, 2017.

[66] H. Miyamoto, S. Schaal, F. Gandolfo, H. Gomi, Y. Koike, R. Osu, E. Nakano, Y. Wada, and M. Kawato, "A Kendama learning robot based on bi-directional theory," *Neural Networks*, Four Major Hypotheses in Neuroscience, vol. 9, no. 8, pp. 1281–1302, 1996.

[67] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[68] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[69] T. Morimura, E. Uchibe, J. Yoshimoto, J. Peters, and K. Doya, "Derivatives of logarithmic stationary distributions for policy gradient reinforcement learning," *Neural computation*, vol. 22, no. 2, pp. 342–376, 2010.

[70] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[71] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and Efficient Off-Policy Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2016.

[72] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," in *Advances in Neural Information Processing Systems*, 2018, pp. 9191–9200.

[73] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning.," in *International Conference on Machine Learning*, 2000.

[74] C. Nota and P. S. Thomas, "Is the policy gradient a gradient?" *arXiv preprint arXiv:1906.07073*, 2019.

[75] Y. Pan, M. Zaheer, A. White, A. Patterson, and M. White, "Organizing experience: A deeper look at replay mechanisms for sample-based planning in continuous state domains," in *International Joint Conference on Artificial Intelligence*, AAAI Press, 2018, pp. 4794–4800.

[76] G. Papamakarios, I. Murray, and T. Pavlakou, "Masked autoregressive flow for density estimation," in *Advances in Neural Information Processing Systems*, 2017, pp. 2335–2344.

[77] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems*, 2013, pp. 2616–2624.

[78] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell, "Zero-shot visual imitation," in *International Conference on Learning Representations*, 2018.

[79] J. Peng and R. J. Williams, "Efficient learning and planning within the Dyna framework," *Adaptive Behavior*, vol. 1, no. 4, pp. 437–454, 1993.

[80] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics*, vol. 37, no. 4, p. 143, 2018.

[81] J. Peters, K. Mülling, and Y. Altun, "Relative Entropy Policy Search," in *AAAI Conference on Artificial Intelligence*, 2010, pp. 1607–1612.

[82] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[83] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[84] D. a Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989.

[85] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal Difference Models: Model-Free Deep RL for Model-Based Control," in *International Conference on Learning Representations*, 2018.

[86] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[87] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[88] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *International Conference on Machine Learning*, ACM, 2006, pp. 729–736.

[89] S. Reddy, A. D. Dragan, and S. Levine, "Sqil: Imitation learning via regularized behavioral cloning," *arXiv preprint arXiv:1905.11108*, 2019.

[90] D. J. Rezende and F. Viola, "Taming vaes," *arXiv preprint arXiv:1810.00597*, 2018.

[91] S. Ross and J. A. Bagnell, "Efficient Reductions for Imitation Learning," in *International Conference on Artificial Intelligence and Statistics*, 2010.

[92] S. Ross, G. Gordon, and J. A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *International Conference on Artificial Intelligence and Statistics*, 2011.

[93] H. Sahni, T. Buckley, P. Abbeel, and I. Kuzovkin, "Visual Hindsight Experience Replay," *Advances in Neural Information Processing Systems*, 2019.

[94] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal Value Function Approximators," in *International Conference on Machine Learning*, 2015, pp. 1312–1320.

[95] Y. Schroecker, H. Ben Amor, and A. Thomaz, "Directing Policy Search with Interactively Taught Via-Points," in *Autonomous Agents and Multiagent Systems*, Singapore, Singapore: International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[96] Y. Schroecker and C. L. Isbell, "State Aware Imitation Learning," *Advances in Neural Information Processing Systems 30*, no. Nips, pp. 2915–2924, 2017.

[97] Y. Schroecker, M. Vecerik, and J. Scholz, "Generative predecessor models for sample-efficient imitation learning," in *International Conference on Learning Representations*, 2019.

[98] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[99] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[100] P. J. Schweitzer, "Perturbation theory and finite markov chains," *Journal of Applied Probability*, vol. 5, no. 2, pp. 401–413, 1968.

[101] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *International Conference on Robotics and Automation*, 2018, pp. 1134–1141.

[102] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, D. Sander, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[103] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014.

[104] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[105] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *International Conference on Machine Learning*, 2012, pp. 281–288.

[106] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *International Conference on Machine Learning*, 2017.

[107] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 1998.

[108] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *Advances in Neural Information Processing Systems 12*, pp. 1057–1063, 1999.

[109] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup, "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 761–768.

[110] U. Syed, M. Bowling, and R. E. Schapire, "Apprenticeship learning using linear programming," in *International Conference on Machine Learning*, ACM, 2008, pp. 1032–1039.

[111] U. Syed and R. E. Schapire, "A game-theoretic approach to apprenticeship learning," in *Advances in Neural Information Processing Systems*, 2008, pp. 1449–1456.

[112] E. Theodorou, J. Buchli, and S. Schaal, "Learning policy improvements with path integrals," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 828–835.

[113] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," in *International Conference on Robotics and Automation*, IEEE, 2010, pp. 2397–2403.

[114] J. N. Tsitsiklis and B. V. Roy, "Average cost temporal-difference learning," *Automatica*, vol. 35, pp. 1799–1808, 1999.

[115] J. N. Tsitsiklis and B. Van Roy, "On average versus discounted reward temporal-difference learning," *Machine Learning*, vol. 49, no. 2-3, pp. 179–191, 2002.

[116] J. N. Tsitsiklis, B. Vany Roy, and B. V. Roy, "An analysis of reinforcement learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[117] A. Ude, C. Man, M. Riley, and C. G. Atkeson, "Automatic generation of kinematic models for the conversion of human motion capture data into humanoid robot motion," in *International Conference on Humanoid Robots*, 2000.

[118] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7184–7220, 2016.

[119] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[120] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," in *International Conference on Machine Learning*, vol. 48, 2016, pp. 1747–1756, ISBN: 978-1-5108-2900-8.

[121] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[122] Y. Wada, Y. Koike, E. Vatikiotis-Bateson, and M. Kawato, "A computational model for cursive handwriting based on the minimization principle," in *Advances in Neural Information Processing Systems*, 1993, pp. 727–734.

[123] R. Wang, C. Ciliberto, P. V. Amadori, and Y. Demiris, "Random expert distillation: Imitation learning via expert policy support estimation," in *International Conference on Machine Learning*, 2019, pp. 6536–6544.

[124] Z. Wang, J. S. Merel, S. E. Reed, N. de Freitas, G. Wayne, and N. Heess, "Robust imitation of diverse behaviors," in *Advances in Neural Information Processing Systems*, 2017, pp. 5320–5329.

[125] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural evolution strategies," in *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress On*, IEEE, 2008, pp. 3381–3387.

[126] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[127] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and Others, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv preprint arXiv:1802.09564*, 2018.

[128] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, "Modeling interaction via the principle of maximum causal entropy," in *International Conference on International Conference on Machine Learning*, Omnipress, 2010, pp. 1255–1262.

[129] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum Entropy Inverse Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2008.